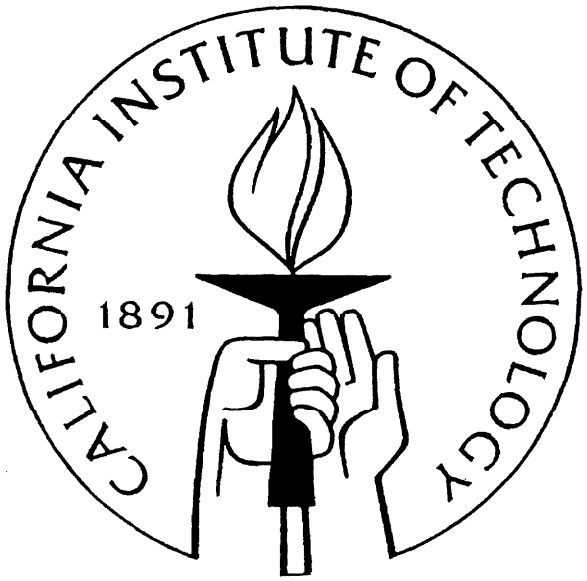

Concurrent Simulations of Plasma Reactors for VLSI Manufacturing

Marc A. Rieffel

Computer Science Department
California Institute of Technology

Caltech-CS-TR-95-12

Acknowledgments

The author would like to thank Sadasivan Shankar for assistance with physics details and industrial guidance, David Weaver for Skipper information and DSMC fundamentals, and Mikhail Ivanov for validation suggestions. Evan Cohn provided Paragon support and optimization advice, Thanh Phung obtained the GEC results on the Paragon, presented in Section 6.2, and Jerrell Watts obtained the load balancing results in Section 6.5 and assisted with ports to the Intel Paragon and Cray T3D. Bradley Nelson wrote *XHawk* and the chemistry database and helped with the descriptions in Sections A.1 and A.2. Nathan Mates generated GEC pictures and movies presented in Chapter 6, and Jeffrey Ho created the plasma diagrams, Figures 1.2 and 1.3. Vince McKoy contributed useful chemistry insights, and Paul Miller of Sandia provided detailed GEC reactor information. Robie Samanta Roy contributed editing suggestions. Lowell Morgan of Kinema Research proved Chlorine rate data, and Wayne Christopher and Diane Poirier of ICEM CFD assisted in grid generation and translation. Xiaolin Zhong at UCLA provided heat transfer data to help with validation, presented in Section 5.5. Johnson Wang from the Aerospace Corporation assisted with the development of the transformation algorithm in Section 4.4. Access to the Intel Paragon was provided by Caltech CACR. Access to a Cray T3D was provided by the NASA Jet Propulsion Laboratory, and was facilitated by Caltech. Stephen Taylor provided advice, support, and encouragement, as research advisor.

Contents

Chapter 1

Introduction

Recent advances in microprocessor performance have been driven primarily by improvements in manufacturing technology. New processes and equipment have paved the way for smaller feature sizes and larger die sizes. These in turn have enabled the production of microprocessors with more transistors, operating at lower voltages and higher clock rates. One of the key pieces of equipment in microelectronics manufacturing is the *plasma reactor*, used in 30 to 40 percent of the processing steps. These reactors use plasmas, or energetic rarefied gases, to remove particles from, and deposit particles on, silicon wafers. Improving the design of these reactors, and the processes that they are used for, will enable the microelectronics industry to make smaller, cheaper, and faster microprocessors.

Design and optimization of plasma reactors has so far been largely empirical. Experiments have been conducted to improve process configurations, but because of the high equipment and operating costs of plasma reactors, detailed parametric studies have been economically impractical. Computer-based simulation of the plasma flow inside a reactor will allow manufacturers to evaluate the viability of different reactor designs before they are implemented. Once a reactor has been installed, simulation results will be useful for studying the effects of different operating conditions, thereby optimizing processing stages.

Figure 1.1 shows the outside of a typical plasma reactor. The grid for such a complex three-dimensional geometry could not be reasonably generated by hand. Automatic grid generation tools are therefore a necessity for realistic reactor simulations.

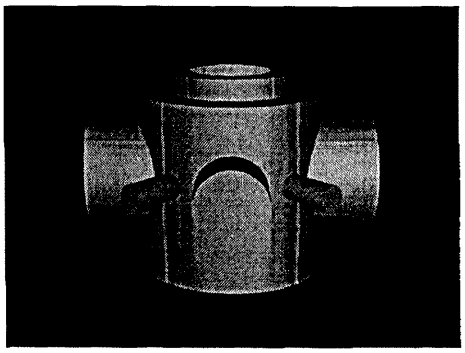

Figure 1.1: A Plasma Reactor

Figure 1.2 schematically depicts the operation of a plasma reactor. A silicon wafer is attached to an electrode, and plasma fills the space between the wafer and another electrode. Gas flows in, reactions take place within the gas and on the surface of the wafer, and the products of these reactions are pumped out of the reactor. Electromagnetic fields, applied through the electrodes, add energy to the system.

The chemical processes in a simple plasma reactor can be described as follows.

1. Gas A flows into the reactor.
2. The endothermic reaction, $A \rightarrow B$, takes place at a heated surface.
3. The reaction products, B, and unconverted reactants, A, are pumped out of the reactor.

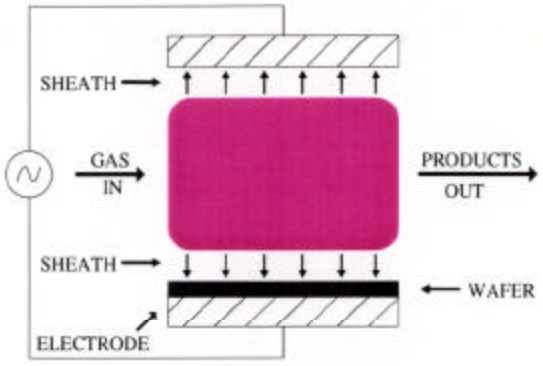

Figure 1.2: Reactor Schematic

Figure 1.3 shows several classes of reactions that take place within a reactor. The most important type of reaction is electron activation, whereby neutral particles are broken up into *free radicals*. Free radicals can also recombine into neutral particles, a process known as deactivation. Ionization takes place when an electron collides with a neutral particle, breaking it into ions. Recombination occurs when electrons combine with ions to form neutrals. On the wafer surface, particles can be implanted through deposition, or ejected through sputtering.

The complexity of plasma simulation is primarily due to the wide range of timescales involved. Figure 1.4 shows the time scales of the processes that are important in plasma reactors. With plasma oscillations taking place in nanoseconds and surface feature evolution spanning seconds, phenomena occuring over 9 orders of magnitude must be addressed. While previous work has been directed at simulating pieces of this problem, inadequate computational resources and insufficient physical models have prohibited comprehensive simulations.

The goal of this project is to use recently-developed high-performance parallel computing technology and sophisticated plasma models to enable complete simulations of plasma reactors. Models of the complete reactor chemistry, self-consistent solution of electromagnetic fields, and support for complex geometries are necessary. In order for simulations to have maximum benefit, simulation turnaround times must be industrially viable. Equipment evaluation typically takes

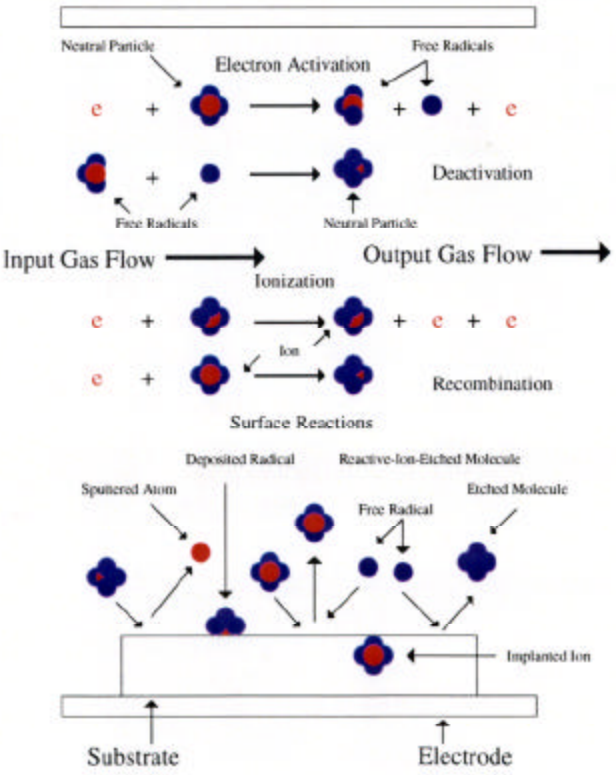

Figure 1.3: Plasma Reactions

The chemical processes in a simple plasma reactor can be described as follows.

1. Gas A flows into the reactor.
2. The endothermic reaction, $A \rightarrow B$, takes place at a heated surface.
3. The reaction products, B, and unconverted reactants, A, are pumped out of the reactor.

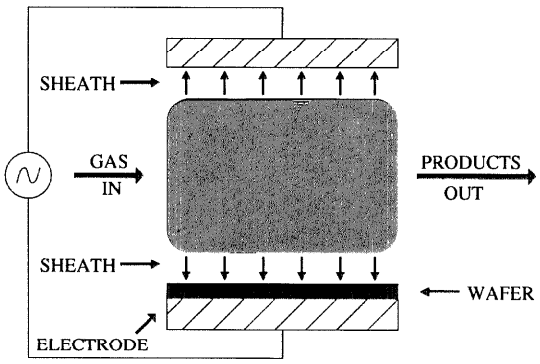

Figure 1.2: Reactor Schematic

Figure 1.3 shows several classes of reactions that take place within a reactor. The most important type of reaction is electron activation, whereby neutral particles are broken up into *free radicals*. Free radicals can also recombine into neutral particles, a process known as deactivation. Ionization takes place when an electron collides with a neutral particle, breaking it into ions. Recombination occurs when electrons combine with ions to form neutrals. On the wafer surface, particles can be implanted through deposition, or ejected through sputtering.

The complexity of plasma simulation is primarily due to the wide range of timescales involved. Figure 1.4 shows the time scales of the processes that are important in plasma reactors. With plasma oscillations taking place in nanoseconds and surface feature evolution spanning seconds, phenomena occuring over 9 orders of magnitude must be addressed. While previous work has been directed at simulating pieces of this problem, inadequate computational resources and insufficient physical models have prohibited comprehensive simulations.

The goal of this project is to use recently-developed high-performance parallel computing technology and sophisticated plasma models to enable complete simulations of plasma reactors. Models of the complete reactor chemistry, self-consistent solution of electromagnetic fields, and support for complex geometries are necessary. In order for simulations to have maximum benefit, simulation turnaround times must be industrially viable. Equipment evaluation typically takes

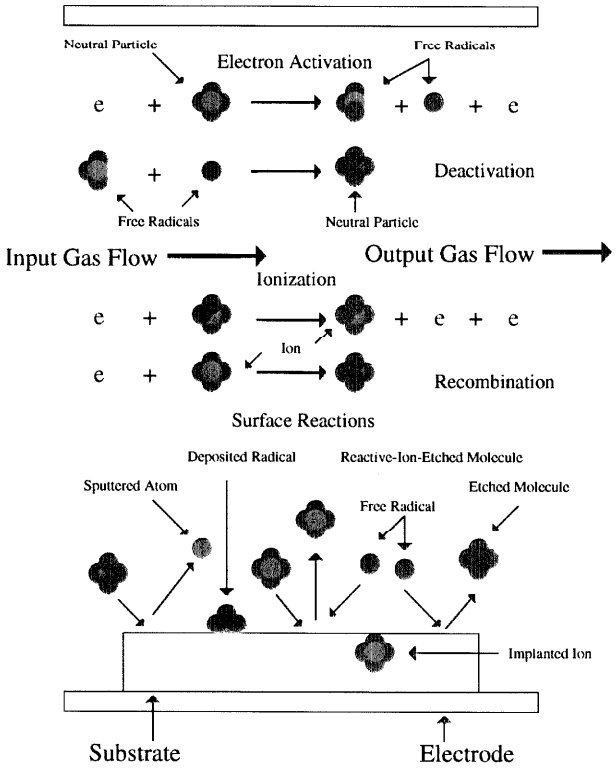

Figure 1.3: Plasma Reactions

place over six months, and process evaluation is conducted in less than a month. In order to be useful for process optimization, simulations must finish overnight.

Parallel computing technology continues to evolve as new machines appear each year, with increasing numbers of processors. To protect the investment in software development, and to take advantage of the most modern equipment, simulation tools must be *portable* and *scalable*. The software must also be *maintainable* and *modular* so that new physics models can be quickly incorporated. These criteria must be met in order for this technology to have impact over a long term.

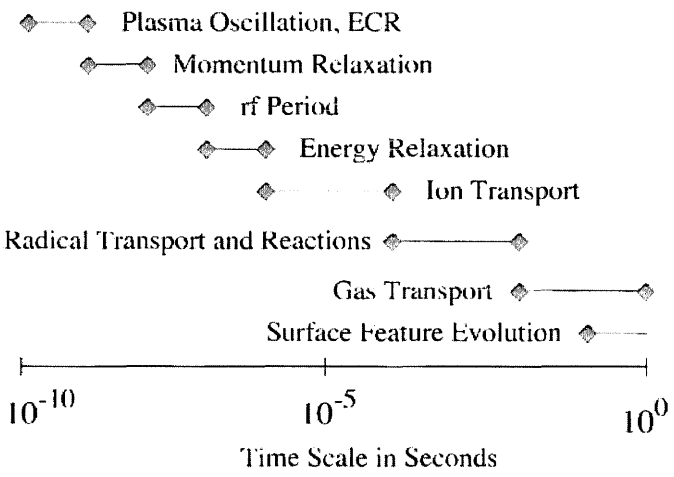

Figure 1.4: Simulation Timescales

Advanced plasma simulation capabilities will be directly applicable to problems in the microelectronics industry and can therefore have direct bearing on industrial competitiveness. Simulations will be useful to study process optimization, compact model development, equipment evaluation, process control, and technology feasibility. Efficient modeling will reduce the time and cost of microelectronics development, and therefore help improve the quality of the next generation of microprocessors.

Computer-aided modeling is intended, not to replace, but to compliment experimental analysis. As with any modeling, some assumptions are necessary to make this approach feasible. Most notable are models of the collision process and gas-surface interactions. The lack of data for these processes imposes a severe constraint on the range of problems that can be solved. In the absence of experimental data, suitably simplified models have been developed. These simplifications, however, may no longer be appropriate when

1. Wafer sizes increase: the feature-size models that have been developed for specific processes will have to be recharacterized for different process settings.

2. Feature sizes decrease: at low pressures, when the inter-particle collision frequency is reduced, surface collisions are increasingly important.

3. Precise process control is necessary: Real-time control and process modeling increase the complexity of interactions within the system. Advanced models will be required for the design of control mechanisms.

Plasma flow is in the *transition regime*. The mean free path of particles is too large for traditional continuum CFD methods to be applicable. Because collisions are important, free-molecular simulations are not appropriate. An alternative approach is to simulate individual particles as they move about within a grid, colliding with solid objects and other particles. Macroscopic properties, such as density and temperature, can be computed by appropriate averaging

of particle masses, positions, and velocities. Surface properties are calculated from the momentum and energy exchanges during collisions with surfaces. This technique, named Direct Simulation Monte Carlo (DSMC), was developed by Bird [Bird94]. The computational requirements of DSMC have, in the past, limited applications to relatively simple geometries or highly rarefied flows. Recently developed massively parallel machines and clusters of high-performance workstations have made it possible to address more complex problems.

This thesis presents *Hawk*, a concurrent implementation of the DSMC technique, that has been used to simulate neutral flow in realistic 3D reactors. It incorporates a unique collection of features that have enabled the first simulations of realistic reactors. Some of the capabilities that distinguish *Hawk* from other plasma simulation tools and DSMC implementations are:

1. **Portability:** In order to benefit from all available computational resources, it was important that *Hawk* run on a variety of platforms. This portablility was accomplished by building the application on top of the *Concurrent Graph Library* . *Hawk* has been built and tested on 512 Intel Paragon processors, 256 Cray T3D processors, 4 SGI Power Challenge processors, and networks of up to 30 workstations (IBM, SGI, and Sun).

2. **Scalability:** *Hawk* has been used to perform large-scale simulations on as few as 2 SGI Power Challenge processors, and as many as 512 Intel Paragon processors. Two features that enhance scalability are *granularity control* and *dynamic load balancing*.

3. **Support for Complex, 3-D geometries:** *Hawk* is a fully three-dimensional code, using boundary-fitted grids. It supports arbitrarily complex geometries, with computational complexity a function of grid size but not of grid complexity.

4. **Multiple Species and Complex Chemistry:** Support for multiple species, inelastic collisions, and velocity-dependent cross sections has been implemented.

5. **Standard Grid Formats:** No special grid-generation tools are required. *Hawk* accepts grids generated by industry-standard grid-generation packages.

6. **Software Engineering:** All of the chemistry and physics models are isolated in separate modules to facilitate the addition of new models as they are developed. *Hawk* is designed to be easily maintained over many years and through many significant changes.

7. **Chemistry Database:** Chemistry information can be loaded from a database of species information and reaction cross sections.

Support for additional features, not yet implemented, has been incorporated into the software infrastructure. Data structures are present to allow the implementation of a self-consistent field solver. Many of the chemistry routines are general enough to support reactions with arbitrary numbers of reactants and products (for example, the spontaneous breakdown of one large particle into three constituents). The addition of surface chemistry models, once they have been developed, will also be straightforward.

Hawk has been used to simulate two three-dimensional plasma reactors, the GEC Reference cell, and a proprietary Intel reactor. These simulations have been performed on grids with about

half a million grid cells, and with 1-3 million particles. These simulations have required multiple gigabytes of RAM, and tens of hours on a 512-node Intel Paragon.

The effect of this work has been to demonstrate that fully three-dimensional solvers, with support for complex geometry, are necessary to characterize the gas flow in industrial plasma reactors. While two dimensional, axisymmetric calculations are much less expensive computationally, they are insufficient for a wide range of important industrial problems.

The following chapters include a brief summary of key related work, an overview of the DSMC method, a description of the implementation, validation of *Hawk* 's features, and the results of large-scale simulations of the GEC reference cell.

Nomenclature

Particles

Particles represent atoms, molecules, ions, and macroscopic particles (e.g. dust). Each particle has a species identifier (specifying its mass and charge), and position and velocity in three dimensions.

Cells

Cells represent regions of space. They are represented as eight three-dimensional points specifying the corners of a hexahedral volume. Each cell has a list of particles that it contains, as well as some statistical properties of those particles (e.g. maximum velocity and cross section). Cell faces must match geometry boundaries, and cell sizes must be chosen to maintain a local Knudsen number of 2-3.

Grid

A grid is the collection of cells used to define the geometry of the problem. Computational cost is roughly proportional to the number of grid cells used.

Faces

Each cell has 12 faces: 6 quadrilateral faces broken into 2 triangular faces each. Each face may have a boundary condition (specular,diffuse, inflow, outflow, or internal/pass-through). All faces except those of type internal/pass-through store boundary data such as energy and sticking fluxes.

Partitions

Partitions are decompositions of the computational domain. Each partition typically contains a large number of adjacent cells. Each partition maintains communication channels with each of its six neighbors.

Cross Section

A collision or reaction between two species has an associated cross section σ that is used to determine the probability that the reaction will occur. The cross section can be thought of as a measure of how close the particles must be to collide. In general, cross sections for a reaction can be functions of the relative velocity between reacting particles and of their internal energy states.

Boundary Condition

Boundary conditions are used to distinguish cell faces on geometry boundaries. Different boundary condition types can be used to specify what happens when particles hit surfaces (reflection or absorbtion) and whether particles are introduced through surfaces (inflow or emission).

Specular

Specular surfaces are perfectly reflecting. Particles hitting specular surfaces are reflected with no change in energy, with angle of incidence equal to angle of reflection.

Diffuse/Accomodating

Particles that hit diffusely reflecting surfaces are reflected with velocities determined solely by the temperature of the surface. That is, pre-reflection velocity information is lost. This may cause an increase or decrease in the particle's energy.

F_N

F_N is the ratio of real to simulated particles; each simulated particle represents F_N real particles.

Random Fractions

As with all Monte Carlo programs, *Hawk* makes extensive use of random numbers. For the purposes of this thesis, all random numbers are random fractions between 0 and 1, typically represented as R. See Section 5.1 for a detailed description of the random number generator.

Dimensions

All variables in *Hawk* are in SI units. Many implementations use dimensionless, normalized parameters exclusively, in order to minimize rounding errors. The normalization process, however, is more complex than is warranted, and an opportunity for introduction of subtle errors. As Bird says [Bird94],

The molecular models are now sufficiently realistic for the simulation results to be compared directly with the measured values. It is therefore desirable for all the variables in the programs to be in a dimensioned form. If dimensionless results are required, the normalization is best applied to the results....

Chapter 2

Related Work

Reactor Simulations

Hitchon, *et. al.*, have implemented a nonstatistical technique for studying the transport of sputtered neutral particles [Parker95]. Their "convective scheme" (CS) has also been used to solve the Boltzmann equation in a cylindrical geometry [Parker94]. Hitchon has performed a detailed study on the sensitivity of simulations to collision techniques [Hitchon94]. Two dimensional results have been obtained for ion densities in plasma chambers of simple geometry, solved self-consistently with the electrostatic potential [Keiter94, Hitchon94a]. Techniques have also been presented [Hitchon91] that permit the results of computationally inexpensive simulations to be extrapolated to obtain results comparable to those from lengthy simulations.

Wadsworth has applied a parallel, three-dimensional code to simulate a simplified, quarter-symmetric, version of the GEC reference cell [Wadsworth95]. This implementation uses ray tracing and grid-based partitioning under PVM, and includes static load balancing. Bartel has performed axisymmetric simulations of plasma reactors, with multiple species and constant electromagnetic fields [Bartel95]. Electron distributions are solved by a separate fluid model.

Nambu and Uchida have combined the DSMC and PIC techniques to handle self-consistent fields, sputtering, and multiple species [Nambu95]. Their technique has been applied to one-dimensional problems, with decoupled neutral flow, ion formation, sputtering, and the transfer of sputtered atoms.

DSMC Implementations

A number of other researchers have implemented DSMC solvers, that vary in the features that they support. Some focus on complex chemistry but do not support realistic geometries. Others have parallel implementations that are not scalable or do not support load balancing.

Bird pioneered the DSMC technique [Bird70a] and wrote two books about it [Bird94, Bird76]. He has studied the breakdown of translational and rotational equilibrium in gaseous expansions, using rough sphere molecules [Bird70], and his code has been used to simulate flow around the Space Shuttle using 700 cells and 8,000 simulated particles [Bird78].

An alternative approach to working with complex 3-D geometries is to separate the grid from the geometry. Ivanov [Ivanov88, Ivanov91], for example, uses a fixed rectangular grid aligned with the axes, and stores the geometry surface information separately. Particle transport is performed with respect to the surfaces, and collisions and statistics are performed in the cells without regard to the geometry.

Advanced Chemistry Models

Nambu presents simulations of hypersonic flow around a disk, using regular multi-block grids [Nambu89]. Koura has calculated coefficents for the variable soft sphere (VSS) model for air species [Koura92]. This group has also performed simulations of force and heat transfer for flow around a disc, and shown good agreement with experimental data [Legge90].

Marriott and Bartel have compared two sophisticated chemistry techniques, Maximum Entropy (ME) and Borgnakke-Larsen variants (BL), on 2-D axisymmetric grids [Marriott95]. Their results show that the two approaches yield significantly different solutions. Boyd has developed models for energy transfer between vibrational and translational modes, simulating flow over a two-dimensional wedge [Boyd91]. He has also studied the effects of rotational degrees of freedom for jets of iodine vapor impinging on blunt bodies [Boyd94]. These results agree well with experimental data for moderate temperatures (100-500K), but less satisfactorily for high temperatures.

Software Engineering

Yokokawa, *et. al.*, have developed a parallel DSMC implementation using grid partitioning [Yokokawa91]. They present two-dimensional results obtained on a Fujitsu AP1000, and have demonstrated a speedup of up to 42 on 64 processors.

Parsons has written about software engineering methodology for DSMC codes, promoting the use of object-oriented and multi-agent systems paradigms [Parsons95]. Results for two-dimensional object-oriented implementations have been presented. The organization and structure of this code appear to be similar to that of *Hawk* . The use of the multi-agent systems paradigm is an intriguing possibility, particularly in the context of parallel processing.

Experiments and Comparisons

Economou and Aydil have conducted experiments, monitoring etch rates and surface chemistry in plasma reactors [Economou91, Aydil94, Aydil95]. Zhong and Koura have compared DSMC results with solutions of the Burnett and Navier-Stokes equations, for Couette flow and heat transfer problems [Zhong95]. These results were used for validation of *Hawk* 's physics, in Chapter 5. Wadsworth has also studied [Wadsworth93] slip effects in rarefied gas and compared the results of DSMC and Navier-Stokes methods, with a variety of gas-surface interaction models. These results were compared with experimental data obtained by Alofs [Alofs71].

Chapter 3

Description of the Method

3.1 Overview of the DSMC Method

The DSMC (Direct Simulation Monte Carlo) method was developed to allow for the solution of problems that could not be correctly handled by the Euler or Navier-Stokes methods beacause of a high degree of rarefaction [Bird94]. The rarefaction of a gas system is typically expressed in terms of the *Knudsen number* (Kn), the ratio of the mean free path λ to the characteristic length of the system L:

$$Kn = \frac{\lambda}{L}.$$

Navier-Stokes methods begin to break down around $Kn = 0.1$ and are considered inappropriate beyond $Kn = 0.2$.

For Knudsen numbers above 0.1, such as in plasma reactors, it is preferable to apply a discrete particle method, or Boltzmann equation solver. The *Boltzmann equation* for a simple dilute gas is

$$\frac{\partial}{\partial t}(nf) + \mathbf{c} \cdot \frac{\partial}{\partial \mathbf{r}}(nf) + \mathbf{F} \cdot \frac{\partial}{\partial \mathbf{c}}(nf) = \int_{-\infty}^{\infty} \int_{0}^{4\pi} n^2 (f^* f_1^* - f f_1) c_r \sigma d\Omega d\mathbf{c}_1,$$

where n is the number density, f is the velocity distribution function, $\mathbf{r}$ is the position vector in velocity space, $\mathbf{c}$ is the velocity, $\mathbf{F}$ is the external force per unit mass, and the integral describes the collision process [Bird94]. The main problem with solving the Boltzmann equation directly is the evaluation of the collision integral. Using a quadrature formula to evaluate it would require 3,200,000 evaluations at each grid point, for each possible velocity, and at each timestep. This is clearly impractical for realistic simulations. Particle methods, such as the DSMC technique, avoid this complexity by replacing the density distribution function with a number of "simulation" particles that move and collide [Cercignani94]. While most systems of interest would contain at least Avagadro's Number (10^{23}) particles, current computational resources can cope with no more than about 10^8 particles. A small number of simulation particles is therefore used to represent a large number of real particles. By randomizing the process it is possble to obtain the same results as would be obtained through direct simulation of each physical particle.

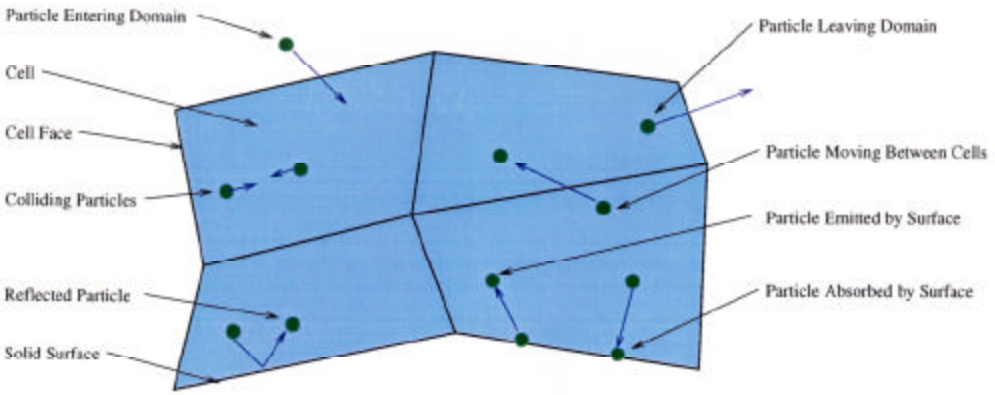

Figure 3.1: DSMC Cells and Particles

In a DSMC simulation, a physical region of interest is decomposed into a number of cells. The cells are initially filled with simulation particles according to density, temperature, and velocity specifications. During each timestep, these particles are allowed to move throughout the domain and collide with other particles. Particles may flow into the domain through injection cells, or out of the domain through exchaust cells. Particles may collide with, or become embedded in, solid objects in the domain. Figure 3.1 shows several two-dimensional cells and some of the possible particle operations.

Macroscopic parameters in a cell, such as density and temperature, can be computed as statistical properties of the particles in the cell. Plots of these parameters can be used to observe trends in the flow, such as pressure gradients, and density variations. The average cell may only have about 10 simulated particles, yielding significant statistical scatter. For steady-state problems, however, smooth results can be obtained by temporally averaging over a large number of timesteps.

As in all Monte Carlo methods, the underlying probability distributions need to be determined. These may be divided into three categories:

1. The distributions of initial positions, velocities, and other particle parameters.
2. The probabilities determining the selection of collision frequency and pairing.
3. The probabilities of energy exchange between internal and external degrees of freedom during the collision process.

Note that collisions, and therefore energy exchanges, can take place between two or more particles, or between a particle and a surface.

In Bird's method, particles enter the computational domain according to the Boltzmann moment equation. Particle velocities are chosen at random from the equilibrium free stream Maxwellian distribution. During one step, each particle moves along a trajectory determined by its position, velocity, and electromagnetic fields, but is uninfluenced by other particles. Collisions during a step are performed after particles have been moved.

The assumptions made in Bird's method, allowing it to be computationally feasible, are as follows:

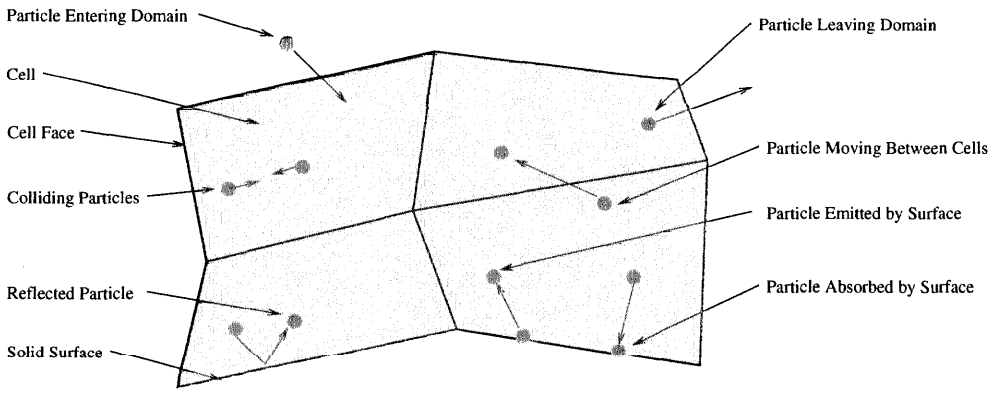

Figure 3.1: DSMC Cells and Particles

In a DSMC simulation, a physical region of interest is decomposed into a number of cells. The cells are initially filled with simulation particles according to density, temperature, and velocity specifications. During each timestep, these particles are allowed to move throughout the domain and collide with other particles. Particles may flow into the domain through injection cells, or out of the domain through exchaust cells. Particles may collide with, or become embedded in, solid objects in the domain. Figure 3.1 shows several two-dimensional cells and some of the possible particle operations.

Macroscopic parameters in a cell, such as density and temperature, can be computed as statistical properties of the particles in the cell. Plots of these parameters can be used to observe trends in the flow, such as pressure gradients, and density variations. The average cell may only have about 10 simulated particles, yielding significant statistical scatter. For steady-state problems, however, smooth results can be obtained by temporally averaging over a large number of timesteps.

As in all Monte Carlo methods, the underlying probability distributions need to be determined. These may be divided into three categories:

1. The distributions of initial positions, velocities, and other particle parameters.
2. The probabilities determining the selection of collision frequency and pairing.
3. The probabilities of energy exchange between internal and external degrees of freedom during the collision process.

Note that collisions, and therefore energy exchanges, can take place between two or more particles, or between a particle and a surface.

In Bird's method, particles enter the computational domain according to the Boltzmann moment equation. Particle velocities are chosen at random from the equilibrium free stream Maxwellian distribution. During one step, each particle moves along a trajectory determined by its position, velocity, and electromagnetic fields, but is uninfluenced by other particles. Collisions during a step are performed after particles have been moved.

The assumptions made in Bird's method, allowing it to be computationally feasible, are as follows:

Figure 3.2: **Sequential DSMC Algorithm**

1. Initialize cells, load particles.
2. Do N steps for each cell
   (a) Inject particles into cell (from inflow or emitting faces).
   (b) Determine the number of collisions in a cell, N_c.
   (c) For each collision:
      i. Randomly select a pair of particles in the cell.
      ii. Determine which reaction will take place between the particles.
      iii. Perform the reaction and compute post-collision velocities.
   (d) Update particle positions using velocities, moving them to new cells if necessary.
   (e) Update particle velocities based on acceleration and fields.

1. Particle movement can be decoupled from particle collisions during a discrete time interval Δt.

2. A small number of computational particles can be used to represent a large number of physical particles. In other words, an average taken over the simulated particles is equivalent, within statistical scatter, to an average taken over all of the real particles.

3. Particles will only collide with other particles in the same cell, and will do so regardless of their positions within the cell.

The DSMC method permits the computation of nonlinear and nonequilibrium flowfields to the extent that physical information is available on the molecular collision process. As far as chemical reactions are concerned, the ideal situation would be to have complete information on the relative cross sections as functions of the geometric impact parameters, the relative translational energy, and the molecular orientations and internal states. The information that is available generally presents averaged cross sections as a function of the relative velocity. However, the cross sections are frequently dependent on the vibrational level of the molecules and additional cross sections may be established for collisional transfers from one level to another. The abstract sequential algorithm of the DSMC method is shown in Figure 3.1. After an initialization phase, a number of timesteps is performed. At each timestep, some number of collisions take place in each cell. After collisions, all particles are moved through the grid with trajectories determined by their positions and velocities, and the electromagnetic fields. The operations of a

timestep in one cell are mostly independent of the states of other cells. Upon completion of the requested number of timesteps, the algorithm terminates.

3.2 Summary of Concepts

While the DSMC algorithm is conceptually simple, a number of details must be addressed at each stage. The primary complexities that must be addressed are particle transport, reactions, and the parallel algorithm.

In most DSMC implementations, particle transport is trivial because it is performed without regard to the grid. That is, particles are moved directly as determined by their positions and velocities and the solid surfaces in the domain. While this approach is very efficient for particle transport, it does not provide information about which particles are contained in a cell. An *indexing* procedure is then necessary to associate particles with cells. For complex grids, the indexing procedure can be computationally intensive. Another disadvantage of this approach is that it does not parallelize well. Information about each solid object in the domain must be stored on every processor. The alternative approach is to use ray tracing for particle transport, associating a particle with a cell during each stage of its movement. This increases the cost of particle transport, but obviates the need for indexing. It also provides convenient opportunities for handling boundary conditions and particle communication. It is easily parallelizable because it requires only local grid information, and it works equally well for automatically-generated unstructured grids.

Reactions are represented as a pair of reactant species, a pair of product species, an energy of reaction, and a velocity-dependent cross section. The probability of the reaction taking place is determined by the reaction cross section. Collisions are treated as a subset of reactions, in which the product species are the same as the reactant species.

Parallelization of the algorithm is achieved by decomposing the grid into partitions. Several partitions may be mapped to each processor, but each partition operates exclusively on its cells and the particles contained therein. Communication is only necessary when particles move between partitions.

Chapter 4

Implementation

This chapter describes the implementations of some of the most distinctive and fundamental features in *Hawk* . The development of the concurrent algorithm is explained, and algorithms for particle tracking and collisions are presented. The handling of initial and boundary conditions is described, and the procedure for computing macroscopic results is explained.

4.1 Simulation Overview

Configuring and running a simulation requires several steps, as shown in Figure 4.1. The starting point is CAD data or engineering drawings describing the geometry to be simulated. This information is used by an automatic grid generation program, such as ICEM-CFD, to create a boundary-fitted grid. While creating the grid, the user specifies which cells have boundary information (in this case, i, o, s and w). Since the parallel algorithm is based on grid-partitioning, the grid must be decomposed into partitions for execution on multiple processors. Cells that are located on the boundaries between partitions are given a new surface type, "Cut".

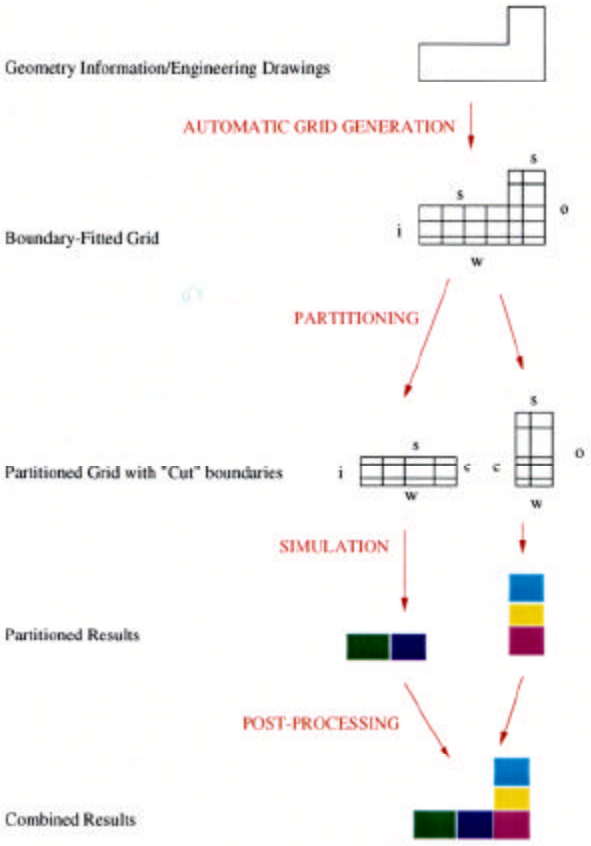

Figure 4.1: Running a Simulation

Once the grid has been partitioned, it can be used in a simulation, along with appropriate configuration information. Once the simulation has terminated, each partition writes its results to a file. A post-processing program is then used

Chapter 4

Implementation

This chapter describes the implementations of some of the most distinctive and fundamental features in *Hawk*. The development of the concurrent algorithm is explained, and algorithms for particle tracking and collisions are presented. The handling of initial and boundary conditions is described, and the procedure for computing macroscopic results is explained.

4.1 Simulation Overview

Configuring and running a simulation requires several steps, as shown in Figure 4.1. The starting point is CAD data or engineering drawings describing the geometry to be simulated. This information is used by an automatic grid generation program, such as ICEM-CFD, to create a boundary-fitted grid. While creating the grid, the user specifies which cells have boundary information (in this case, i, o, s and w). Since the parallel algorithm is based on grid-partitioning, the grid must be decomposed into partitions for execution on multiple processors. Cells that are located on the boundaries between partitions are given a new surface type, "Cut".

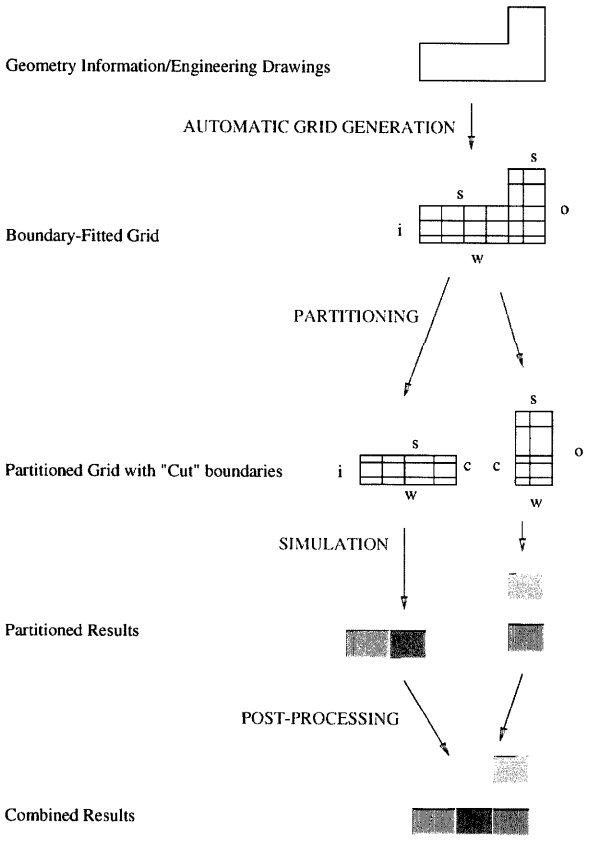

Figure 4.1: Running a Simulation

Once the grid has been partitioned, it can be used in a simulation, along with appropriate configuration information. Once the simulation has terminated, each partition writes its results to a file. A post-processing program is then used

to combine the results into a global results file that can be used for analysis under standard visualization programs such as Visual3 and Data Visualizer.

4.2 Concurrent Algorithm

The grid-partitioning algorithm divides the grid into a number of partitions. A partition is represented by a set of grid cells and the particles that they contain. This allows the memory requirements for each partition to be inversely proportional to the number of partitions. Collisions among particles, and between particles and walls are performed locally. Most particle movement is within a partition and can also be performed locally. The only operation that requires communication between partitions is particle movement from a cell in one partition to an adjacent cell in a neighboring partition.

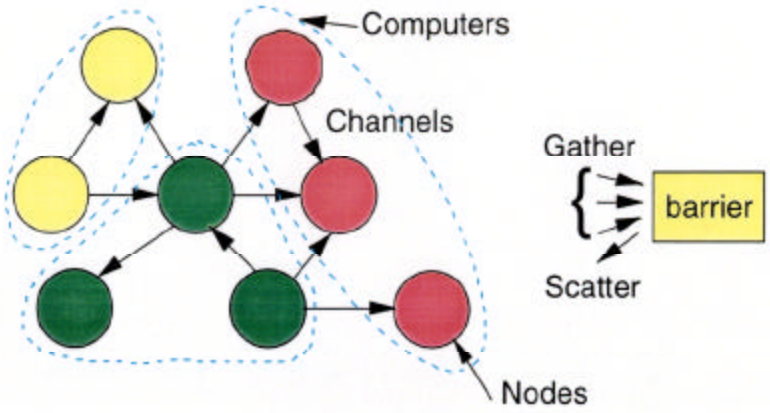

Figure 4.2: The Graph Structure

Hawk 's concurrent structure is described in terms of a graph, consisting of nodes and edges, as shown in Figure 4.2. Each node represents a partition of the grid, and each edge corresponds to a set of cell faces shared between neighboring partitions. A barrier operation is used for gather/scatter operations such as computation of the total number of particles in the system. This approach separates the logical structure of the application from the configuration of the underlying machine, allowing the logical structure to change during program execution. Multiple partitions may be mapped to a single computer in order to overlap communication and computation. Setting the number of cells per partition also facilitates control over the granularity of the computation, i.e. the ratio of computation to communication. The granularity can even be dynamically adjusted by splitting and combining partitions, and dynamic load balancing can be achieved by moving partitions between computers [Watts95]. (See Section 6.5 for load balancing results.)

The parallel algorithm is almost identical to the sequential algorithm. In both, each cell moves and collides its particles independently. During the course of a timestep in a partition, particles move across cell faces on the ends of the partition. These particles must be sent to the appropriate neighboring partitions. Each partition maintains a list of particles to be sent to each of its neighbors. When a particle moves out of a partition, it is added to the appropriate send list. At the end of a timestep, each of these lists is packed into a separate partition of memory and sent as a message to the appropriate neighbor. Each partition therefore sends and receives six messages at the end of a timestep.

When a particle crosses a cell face that is a boundary between two partitions, it generally

to combine the results into a global results file that can be used for analysis under standard visualization programs such as Visual3 and Data Visualizer.

4.2 Concurrent Algorithm

The grid-partitioning algorithm divides the grid into a number of partitions. A partition is represented by a set of grid cells and the particles that they contain. This allows the memory requirements for each partition to be inversely proportional to the number of partitions. Collisions among particles, and between particles and walls are performed locally. Most particle movement is within a partition and can also be performed locally. The only operation that requires communication between partitions is particle movement from a cell in one partition to an adjacent cell in a neighboring partition.

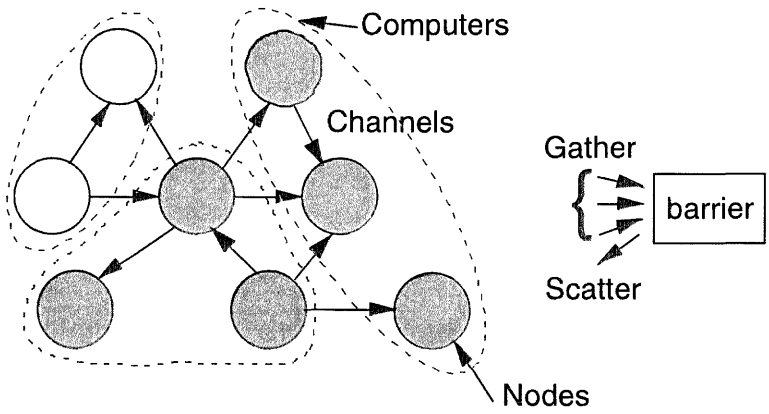

Figure 4.2: The Graph Structure

Hawk 's concurrent structure is described in terms of a graph, consisting of nodes and edges, as shown in Figure 4.2. Each node represents a partition of the grid, and each edge corresponds to a set of cell faces shared between neighboring partitions. A barrier operation is used for gather/scatter operations such as computation of the total number of particles in the system. This approach separates the logical structure of the application from the configuration of the underlying machine, allowing the logical structure to change during program execution. Multiple partitions may be mapped to a single computer in order to overlap communication and computation. Setting the number of cells per partition also facilitates control over the granularity of the computation, i.e. the ratio of computation to communication. The granularity can even be dynamically adjusted by splitting and combining partitions, and dynamic load balancing can be achieved by moving partitions between computers [Watts95]. (See Section 6.5 for load balancing results.)

The parallel algorithm is almost identical to the sequential algorithm. In both, each cell moves and collides its particles independently. During the course of a timestep in a partition, particles move across cell faces on the ends of the partition. These particles must be sent to the appropriate neighboring partitions. Each partition maintains a list of particles to be sent to each of its neighbors. When a particle moves out of a partition, it is added to the appropriate send list. At the end of a timestep, each of these lists is packed into a separate partition of memory and sent as a message to the appropriate neighbor. Each partition therefore sends and receives six messages at the end of a timestep.

When a particle crosses a cell face that is a boundary between two partitions, it generally

Figure 4.3: **Parallel Algorithm**

1. Initialize All Partitions
2. While more steps are appropriate
   (a) Prepare partitions for timestep
   (b) While there are still particles to be exchanged
      - Perform a round of exchanging messages (particle lists)
   (c) Execute a timestep in parallel
   (d) Chekpoint problem state periodically
3. Conclude computation

does so in the middle of the path that it will follow during its current timestep. That is, the time it takes the particle to go from its starting position to the cell face is less than the timestep during which the particle must move. It must therefore continue to move after crossing the boundary cell face. This movement, however, cannot be performed until after the particle has arrived in its new partition. For this reason, particles traveling between partitions keep track of their *remaining time*: the length of time during which they must be moved *after* arriving in their destination partitions.

Each cell stores the locations of its neighbor cells. When a particle leaves a cell through a cut, its destination cell is therefore easily determined. When a partition receives a message from one of its neighbors, it unpacks the message into a list of particles. It then inserts each of the particles into its destination cell, and moves it for the appropriate remaining time. In some cases, this continued movement will cause particles to leave their new partitions. In this case, another round of message passing is required.

During each timestep, a partition computes aggregate statistics about its constitutent cells, such as the total number of particles contained, number flowing in and out of the domain, and number of particles communicated. These parameters are written to a log file to allow the user to monitor the progress of the simulation.

4.3 Portability

Portability of *Hawk* 's communication and synchronization routines is obtained through use of the *Concurrent Graph Library* [Taylor95]. All *Concurrent Graph Library* applications share the same structure, allowing the functionality to be encapsulated in consistent software interfaces. Communication in the *Concurrent Graph Library* is based on a low latency remote-

procedure call (RPC). This mechanism represents the lowest common denominator of available programming models. It can be implemented with a variety of methods, including pointer copying on shared-memory machines and message-passing on distributed systems. Hardware implementations are also available in experimental architectures [Dally94, Maskit94]. By focusing on a single communication concept, portable applications are easier to construct and maintain. The library is in use on a wide range of multicomputers, shared-memory multiprocessors, and networked workstations, including the Cray T3D, Intel Paragon, IBM SP2, and SGI Power Challenge.

4.4 Particle Transport

For structured grids, there are two methods for determining which cell a particle is located in. One is based on a coordinate transformation, and the other on ray tracing.

Transport using Coordinate Transformations

The coordinate transformation method of particle transport assumes a structured hexahedral grid. A structured hexahedral grid is essentially a mapping of points from computational space (ξ, η, ζ) to physical space (x, y, z). Figures 4.4 and 4.5 show a 3-D grid in physical and computational space, respectively. At each of these points, it is possible to use central-differences to compute partial derivatives of (x, y, z) with respect to (ξ, η, ζ). This yields the matrix of transformation metrics [Anderson84],

$$\begin{pmatrix} x_\xi & y_\xi & z_\xi \\ x_\eta & y_\eta & z_\eta \\ x_\zeta & y_\zeta & z_\zeta \end{pmatrix},$$

where each of these parameters involves a partial derivative with respect to the computational axes. Since transformation information is only available at discrete points, it is necessary to compute these derivatives by means of central differences. To compute the parameter x_η, for example, the derivative $\frac{\partial x}{\partial \eta}$ is taken. Given three points along the x direction, 1, 2, and 3, x_η at point 2 can be computed using

$$x_\eta \equiv \frac{\partial x}{\partial \eta} \approx \frac{x_3 - x_1}{\eta_3 - \eta_1}$$

The computational space, however, is uniformly spaced and dimensionless. Each grid cell can therefore be designated a unit cube in computational space, and the denominator of the derivative can then be reduced from $\eta_3 - \eta_1$ to $3-1 = 2$. For any interior grid point with indices (ξ, η, ζ) it is possible to compute x_η using

$$x_\eta(\xi, \eta, \zeta) = \frac{x(\xi + 1, \eta, \zeta) - x(\xi - 1, \eta, \zeta)}{2}.$$

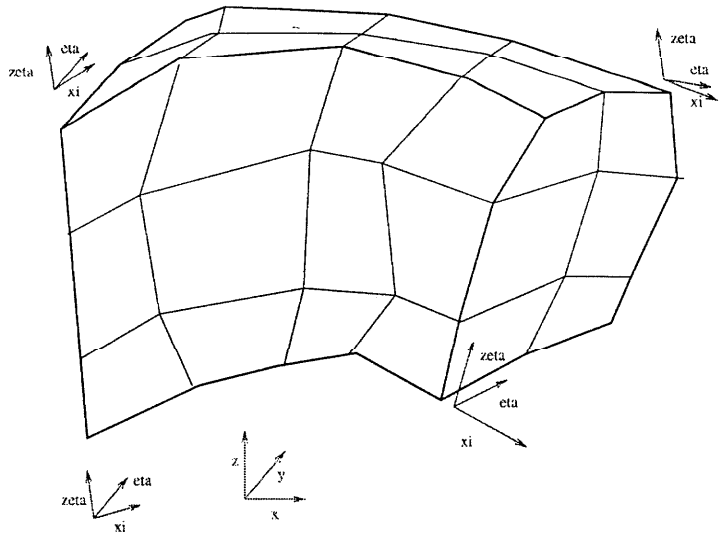

Figure 4.4: Physical Space

The other metrics x_ξ, y_ξ, etc., are computed similarly. For particle transport, however, the inverses of these parameters are necessary, in other words, the elements of the matrix,

$$M = \begin{pmatrix} \xi_x & \eta_x & \zeta_x \\ \xi_y & \eta_y & \zeta_y \\ \xi_z & \eta_z & \zeta_z \end{pmatrix}.$$

To find these, one must first compute the Jacobian J of the transformation at each grid point [Hildebrand76],

$$J = \begin{vmatrix} \xi_x & \eta_x & \zeta_x \\ \xi_y & \eta_y & \zeta_y \\ \xi_z & \eta_z & \zeta_z \end{vmatrix} = 1/\begin{vmatrix} x_\xi & y_\xi & z_\xi \\ x_\eta & y_\eta & z_\eta \\ x_\zeta & y_\zeta & z_\zeta \end{vmatrix}.$$

To transform the point $\vec{p}$ from physical space to computational space, the functions $\xi(x,y,z)$, $\eta(x,y,z)$, and $\zeta(x,y,z)$ are necessary. Since these functions are only precisely defined at the grid points, a Taylor expansion is necessary to find their values elsewhere. Given that $\vec{p}$ is in the grid cell with indexes (ξ,η,ζ), the expansion can be performed about a point $\vec{o}$ in the center of that grid cell. While the values of x, y, z, ξ,η,ζ, and the transformation parameters ξ_x, etc., at $\vec{o}$, are unknown, they can be approximated as lying between the values at the grid point with indices (ξ,η,ζ) and the one with indices $(\xi+1,\eta+1,\zeta+1)$.

A Taylor expansion can be performed about $\vec{o}$ to obtain $\vec{p}'$, the transformation of $\vec{p}$. The expansion equation is,

$$\vec{p}' = \vec{o}' + M(\vec{p} - \vec{o}),$$

where M is the matrix of transformation parameters and $\vec{o}'$ is the transform of $\vec{o}$. If a particle starts at position $\vec{p_0}$ with grid coordinates (ξ_0,η_0,ζ_0), its position $\vec{p_1}$ after translating with velocity $\vec{v}$ during a timestep Δt is given by,

$$\vec{p_1} = \vec{p_0} + \vec{v}\Delta t.$$

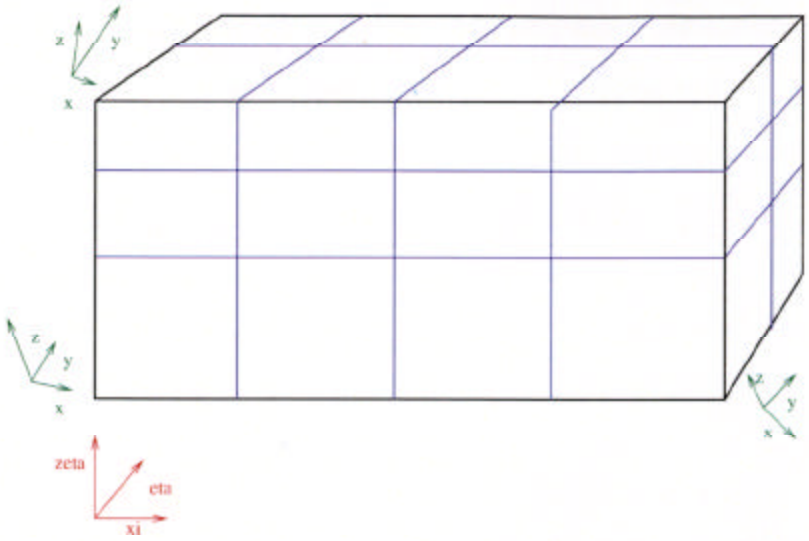

Figure 4.5: Computational Space

Its new grid coordinates, (ξ_1, η_1, ζ_1) are computed using the technique described above. If (ξ_1, η_1, ζ_1) is different from (ξ_0, η_0, ζ_0), the particle has moved to a new cell.

Transport Using Ray Tracing

The ray tracing method of particle transport makes no assumption about the connectivity of a grid, but it requires that cell faces be planar. Each of the quadrilateral faces of a hexahedral cell must therefore be split into two triangular faces. In tetrahedral grids, cell faces are already triangular, thus no splitting is necessary.

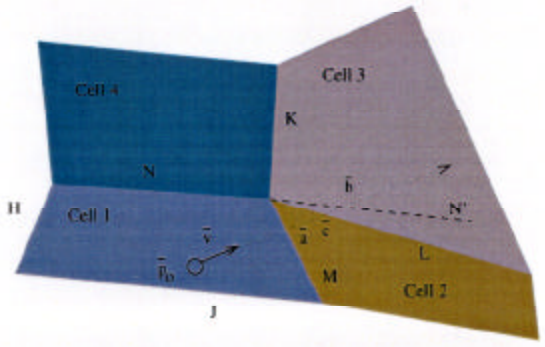

Figure 4.6: Ray Tracing Transport

Figure 4.6 shows a two-dimensional (for the sake of simplicity) representation of a sample geometry. Cells are represented as quadrilaterals and cell faces as line segments. Consider a particle at position $\vec{p_0}$ with velocity $\vec{v}$, initially located in Cell 1. The particle's vector position and cell location at the end of the timestep Δt are determined by its position, velocity, starting cell, and a timestep length.

To determine whether a particle will move into another cell, it is necessary to calculate which cell face the particle will hit first. Consider each of the cell faces (H,J,M,N) of the starting cell. By computing the dot product between the face normals and the particle's velocity, it is possible to determine which cell faces define planes that the particle's trajectory will intersect, thus identifying a set of candidate faces. For each of these, the point of intersection between the particle's trajectory and the plane containing the cell face, as well as the time that it will take the particle

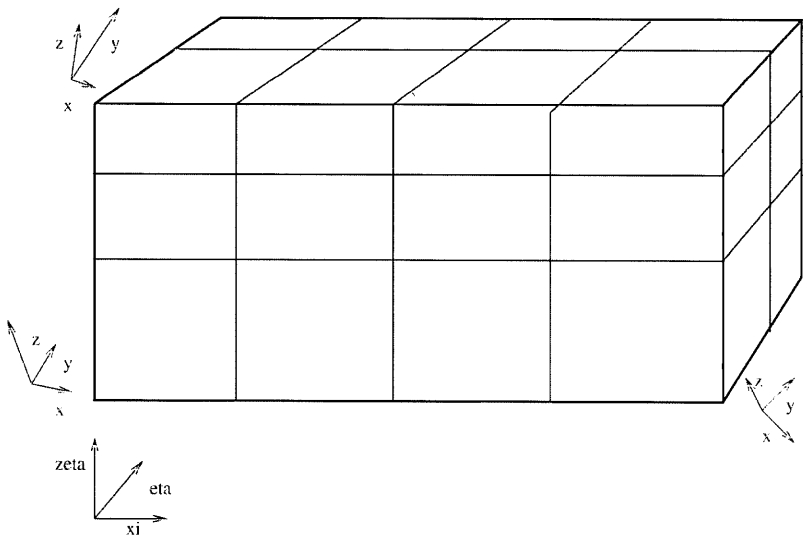

Figure 4.5: Computational Space

Its new grid coordinates, (ξ_1, η_1, ζ_1) are computed using the technique described above. If (ξ_1, η_1, ζ_1) is different from (ξ_0, η_0, ζ_0), the particle has moved to a new cell.

Transport Using Ray Tracing

The ray tracing method of particle transport makes no assumption about the connectivity of a grid, but it requires that cell faces be planar. Each of the quadrilateral faces of a hexahedral cell must therefore be split into two triangular faces. In tetrahedral grids, cell faces are already triangular, thus no splitting is necessary.

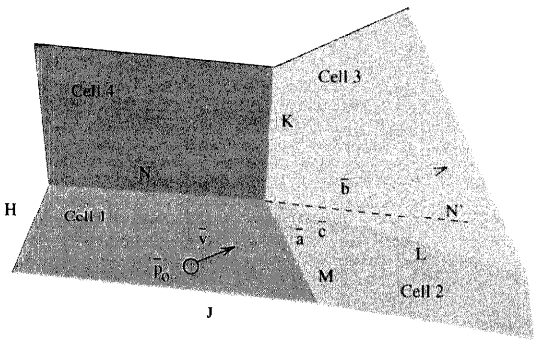

Figure 4.6: Ray Tracing Transport

Figure 4.6 shows a two-dimensional (for the sake of simplicity) representation of a sample geometry. Cells are represented as quadrilaterals and cell faces as line segments. Consider a particle at position $\vec{p_0}$ with velocity $\vec{v}$, initially located in Cell 1. The particle's vector position and cell location at the end of the timestep Δt are determined by its position, velocity, starting cell, and a timestep length.

To determine whether a particle will move into another cell, it is necessary to calculate which cell face the particle will hit first. Consider each of the cell faces (H,J,M,N) of the starting cell. By computing the dot product between the face normals and the particle's velocity, it is possible to determine which cell faces define planes that the particle's trajectory will intersect, thus identifying a set of candidate faces. For each of these, the point of intersection between the particle's trajectory and the plane containing the cell face, as well as the time that it will take the particle

to reach this point, are calculated.

Any plane can be described by the equation

$$Ax + By + Cz + D = 0,$$

where (x, y, z) are Cartesian coordinates and A, B, C and D are parameters specific to the plane. The equation for the position $\vec{p}$ of the particle at any time t is,

$$\vec{p} = \vec{p_0} + \vec{v}t,$$

or

$$x = p_x + v_x t$$

$$y = p_y + v_y t$$

$$z = p_z + v_z t.$$

Substituting the particle equations into the plane equation, it is possible to solve for the time t_c that it takes for the particle to reach the plane,

$$t_c = \frac{Ap_x + Bp_y + Cp_z + D}{Av_x + Bv_y + Cv_z}.$$

If the denominator is equal to zero, the particle's trajectory is parallel to the plane, so there will be no intersection. If the ratio is negative, the particle is moving away from the plane and no intersection is found [Foley94].

The parameters A, B, C are conveniently specified as the coordinates of the surface normal vector, $\hat{n}$, of the plane. The parameter D can be written as the dot product of $\hat{n}$ and any point $\vec{r}$ in the plane. This equation can then be simplified to,

$$t_c = \frac{\hat{n} \cdot \vec{p_0} + \hat{n} \cdot \vec{r}}{\vec{v} \cdot \hat{n}}.$$

Substituting t_c into the equation of motion for the particle will give the point of intersection, $\vec{r}$,

$$\vec{r} = \vec{p_0} + \vec{v}t_c.$$

In Figure 4.6, the intersection between the particle's trajectory and the plane containing face M is point $\vec{a}$, while the intersection with the plane containing face N (N') is the point $\vec{b}$. Since $\vec{p_0}$ is closer to $\vec{a}$ than to $\vec{b}$, the time to intersection with face M is less than the time to intersection with face N. The face with the shortest intersection time will be the first one to be hit by the particle.

Once the closest face has been found, its corresponding intersection time is compared to the timestep length Δt. If the intersection time is greater than Δt, the particle will not hit any faces in the current timestep. It can therefore be moved directly to its new position

$$\vec{p_1} = \vec{p_0} + \vec{v}\Delta t.$$

If, however, the intersection time is less than the timestep, the particle will hit the corresponding cell face.

When the intersection point is found, the particle is moved there. The future of the particle then depends on the nature of the face that it has hit. If the face is solid, the particle will be reflected. If it an outflow surface, the particle is ejected from the simulation. If it is a boundary between partitions (cut), the particle is packed into a message for communication to a neighboring partition. If it is an internal boundary, the particle is moved to the neighboring cell.

In the case of internal and solid boundaries, further movement is required. This is achieved through a recursive call to the particle movement routine. In other words, the particle is then moved with its updated cell, position, and velocity, for the remainder of the timestep. If the time to collision was t_c, the remaining time is $t_r = \Delta t - t_c$.

Transport Method Comparison

The advantage of the transformation method is that particle movement can be completed in one constant-time step, whereas the ray tracing method requires one step for each cell face crossing, and the cost of this step is proportional to the number of faces in the cell. The transformation method, however, does not provide exact points of intersection between particle trajectories and cell faces. These locations are very important for performing reflection and accomodation calculations.

The transformation method is only applicable to single-block, structured hexahedral grids, while the ray tracing method can also be used on unstructured hexahedral and tetrahedral grids. Transport by ray tracing provides the exact time and location of every cell face crossing, and offers a convenient opportunity for handling boundary conditions. The cost of the method is only a function of the number of cell faces in the grid, not the complexity of the geometry. Finally, this approach is consistent with adaptive meshing strategies.

Early versions of *Hawk* used transformations to determine when particles moved between cells within a partition, and ray tracing to determine when a particle left a partition, and to calculate reflections off of solid boundaries. The combination of approaches, however, allowed for slight numerical inconsistencies that were difficult to resolve. The transformation method was therefore abandoned, and ray tracing is now used for all cell-crossing computations. This eliminated the tedious mathematics used for transformations, and therefore substantially simplified the implementation.

4.5 Chemistry

Chemistry information is used by *Hawk* in several situations. When a particle leaves the domain, its mass is used in computing an energy flux. When particles move under the influence of electromagnetic fields, their masses and charges are used to compute accelerations. Information about reactions is used to determine how many collisions must take place in a cell, to decide which particles will collide, and to compute the effects of the collision.

Species

The data associated with a species, or chemical compund, includes its name, mass, and charge. All of the species used in a reaction are stored in a table. Particles therefore need only store indexes into this table, rather than storing name, mass, and charge individually.

Reactions

All collisions, reactive and non-reactive, elastic and inelastic, are stored in the same generic form,

$$A + B \rightarrow C + D + \xi,$$

where A and B are the reactants, C and D are the products, and ξ is the net energy of the reaction. A, B, C, and D are indexes into the species table described above. ξ is the net energy of the reaction. The cross section of the reaction as a function of relative velocity, $\sigma(v_r)$, is stored in the form

$$\sigma(v_r) = p_0 + p_1 v_r + p_2 v_r^2 + p_3 v_r^3 + e_m \exp(e_r v_r),$$

where p_0-p_3 are polynomial coefficients, and e_m and e_r are exponential coefficients.

Reactions are stored in a table, with all of the reactions involving the same products grouped together. This allows for quick retrieval of all of the possible reactions between a pair of particles once they have been selected for a possible collision. Figure 4.5 shows some of the reactions that would be relevant to a chlorine reactor. Cross sections for these reactions are typically available in the form of rate constants, which can be converted to cross sections in the form described above.

4.6 Collisions

Collisions are arguably the most important operation in the DSMC technique. The two central questions are: a) selection: which pairs of particles in a cell will collide, and b) collision: what happens when two particles collide. In this implementation, collisions are a subset of reactions. That is, collisions are reactions in which the products are the same as the reactants, and there is no net energy of reaction.

Number of Collisions

The first step in the collision process is to determine the maximum number of possible collisions. The collision rate per particle, or collisions per particle per unit time, R_m, is given by

$$R_m = n\overline{v_r \sigma(v_r)},$$

where n is the number density of particles, v_r is the relative velocity between a colliding pair, $\sigma(v_r)$ is the cross section of the collision between a colliding pair, and $\overline{v_r \sigma(v_r)}$ is the average

Figure 4.7: **Chlorine Reactions**

Electron Reactions

$Cl_2 + e \rightarrow 2Cl + e$ (dissociation)
$Cl_2 + e \rightarrow Cl_2^+ + 2e$ (ionization)
$Cl + e \rightarrow Cl^+ + 2e$ (ionizatoin)
$Cl^- + e \rightarrow Cl + 2e$ (detachment)
$Cl_2 + e \rightarrow Cl^- + Cl$ (dissociation attachment)
$Cl + e \leftrightarrow Cl^+ + 2e$ (ionization)

Molecular Reactions

$Cl^{2+} + Cl^- \rightarrow Cl_2 + Cl$ (mutual neutralization)
$Cl^+ + Cl^- \rightarrow Cl_2$ (mutual neutralization)
$Cl^+ + Cl_2 \rightarrow Cl_2^+ + Cl$ (charge transfer)
$2Cl \rightarrow Cl_2$ (diffusion)

product of relative velocity and cross section. In a single-species system, the number of collisions per unit volume per unit time, R_c, can be written,

$$R_c = \frac{1}{2} n^2 \overline{v_r \sigma(v_r)}.$$

Rewriting the number density n as the ratio of particles in the cell, N_{cell}, to the volume of the cell, V_{cell}, yields

$$R_c = \frac{N_{cell}^2}{2V_{cell}^2} \overline{v_r \sigma(v_r)}.$$

The number of real collisions C_r in a cell during a timestep of length Δt can then be obtained by multiplying R_c by Δt and the cell volume V_{cell},

$$C_r = \frac{N_{cell}^2}{2V_{cell}} \overline{v_r \sigma(v_r)} \Delta t.$$

Since each simulation particle represents a large number of physical particles, the number of simulated collisions, C_s is obtained by dividing this by F_N,

$$C_s = \frac{N_{cell}^2 \overline{v_r \sigma(v_r)} \Delta t}{2V_{cell} F_N}.$$

The quantity $\overline{v_r \sigma(v_r)}$ is given by

$$\overline{v_r \sigma(v_r)} = \int v_r \sigma(v_r) f(v_r) dv_r,$$

where $f(v_r)$ is the relative velocity distribution function.

The distribution function f, however, cannot be known directly. The quantity $\overline{v_r\sigma(v_r)}$ is an average over a (generally) nonequilibrium distribution function, and will most likely change in time. The number of simulated collisions is therefore computed using an acceptance - rejection scheme based on the maximum value of $v_r\sigma(v_r)$. For this approach, it is necessary to keep track of the value $[v_r\sigma(v_r)]_{max}$, which is updated with every collisional event. The maximum number of collisions to be considered becomes $\frac{1}{2}N\ \langle n\rangle[v_r\sigma(v_r)]_{max}\Delta t$ where N is the number of simulated particles, and $\langle n\rangle$ is the average number density.

Given a desired number of collision pairs, individual particles are selected for collision. Random indexes into the list of particles are computed, subject to the restriction that no particle can collide with itself. Once possible collision pairs have been selected, each one must be tested for a real collision. Given two potentially-colliding particles, the set of possible reactions between those species is extracted from the reaction table. Each of these reactions has a corresponding cross section $\sigma(v_r)$ and net energy of reaction.

Consider two colliding particles, A and B. The relative velocity between A and B is computed using the equation $v_r = |\vec{v_A} - \vec{v_B}|$. If there are K possible reactions between these particles, the probability that particle A will collide with particle B via reaction k is,

$$p_k = \frac{v_r\sigma_k(v_r)}{(v_r\sigma_k(v_r))_{max}} \frac{\sigma_k(v_r)}{\sum_{j=1}^{K}\sigma_j(v_r)}.$$

The *expected number* of collision events is therefore equal to the product of the number of tests and the probability of each test resulting in a collision,

$$\overline{N_c} = [\frac{1}{2}N\langle n\rangle[v_r\sigma(v_r)]_{max}\Delta t][\frac{v_r\sigma(v_r)}{[v_r\sigma(v_r)]_{max}}] = \frac{1}{2}N\langle n\rangle v_r\sigma(v_r)\Delta t.$$

Hawk implements a slight modification of this procedure based on Ivanov's work [Ivanov88, Ivanov91]. This method assumes an exponential distribution of the number of collisions, with collision frequency

$$f_c = \frac{N_{cell}^2 F_N[v_r\sigma(v_r)]_{max}}{2V_{cell}}.$$

To sample from this distribution, a loop is executed, each time incrementing a value τ by $-\log(\frac{R_f}{f_c})$, where R_f is a random fraction. Once τ exceeds the timestep Δt, the loop terminates, and the number of iterations is taken as the number of times the loop executed. While this modification provides identical results for large numbers of particles per cell, it provides significantly more accurate results for fewer than 10 particles per cell. In fact, this technique has allowed *Hawk* to accurately reproduce experimental results with as few as one particle per cell, as described in Section 5.6.

Because two particles may react in any number of ways, a few more steps are necessary in the reaction selection procedure. First, the maximum number of possible collisions, N_{max}, is selected using,

$$N_{max} = \frac{n}{2}F_N N_{cell}\Delta t[v_r\sigma_k(v_r)]_{max} = \frac{1}{2}F_N\frac{N_{cell}^2}{V_{cell}}[v_r\sigma_k(v_r)]_{max}\Delta t.$$

Figure 4.8: **Collision Loop**

1. Pick a pair of particles.
2. Compute v_r, their relative velocity.
3. Look up the K reactions between the two species.
4. Compute a random number R.
5. For each k of the K possible reactions,
   (a) Increment the cumulative probability by $p_k = \frac{v_r\sigma_k(v_r)}{(v_r\sigma_k(v_r))_{max}} \frac{\sigma_k(v_r)}{\sum_{j=1}^{K}\sigma_j(v_r)}$.
   (b) If the cumulative probability exceeds R, terminate the loop and perform reaction k.

Once this number has been compted, the following loop is executed that many times, as shown in Figure 4.6.

The total number of collisions actually carried out will be on average,

$$N_{max}\frac{\overline{v_r\sigma(v_r)}}{[v_r\sigma(v_r)]_{max}} = N_c.$$

One way to find $[v_r\sigma(v_r)]_{max}$ is to compute an initial guess, and then to replace it with the current $v_r\sigma(v_r)$ determined from the collision pair under examination whenever this new value is found to be greater than the previous maximum. If the initial guess is too small, too few collisions will be performed until $[v_r\sigma(v_r)]_{max}$ becomes large enough. If the initial guess is too large, more collision tests will be performed than necessary, but the correct number of collisions will be executed.

A reasonable initial guess for $[v_r\sigma(v_r)]_{max}$ in a cell can be obtained from the initial temperature of the cell and knowledge of the collision database. The mean relative speed between particles of mass m in a gas at equilibrium at temperature T is given by

$$\overline{v_r} = \sqrt{\frac{16kT}{m\pi}}.$$

For a given species, the mean relative velocity is computed in this manner. The cross sections for all reactions involving this species are then evaluated at this relative velocity. The maximum value of $v_r\sigma(v_r)$ for all species and all reactions is then computed. This yields an approximation of the maximum product $v_r\sigma(v_r)$.

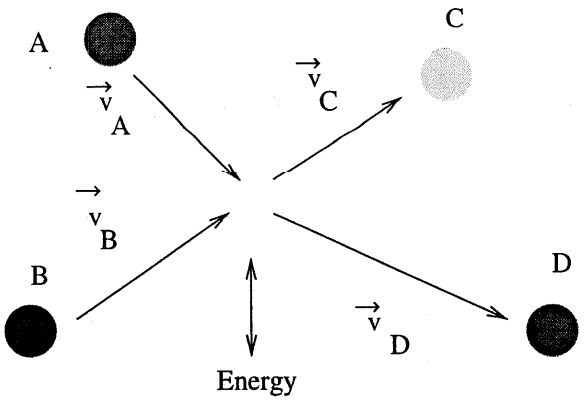

Figure 4.9: Particle Collision

The Collision of Two Particles

Once two particles have been selected for a collision, the collision is performed by modifying the particle species and velocities. Consider two colliding particles as shown in Figure 4.9. Let m_A be the mass of particle A, m_B that of particle B, etc. The velocity of the center of mass of the collision pair is given by,

$$\vec{v_{cm}} = \frac{\vec{v_A} m_A + \vec{v_B} m_B}{m_A + m_B}.$$

The relative velocity between the particles is $\vec{v_r} = \vec{v_B} - \vec{v_A}$, and the relative speed is then $v_r = |\vec{v_r}|$. Let $\hat{e}$ be a unit vector in a random direction, and let $\vec{v_r^*}, \vec{v_C}, \vec{v_D}$ be the post-collision values of the relative velocity, particle A's velocity, and particle B's velocity, respectively.

Random vectors are generated from two random numbers, ϕ (between 0 and 2π) and $\cos\theta$ (between 0 and 1), where θ denotes the angle from the x-axis, and ϕ denotes the angle from the y-axis to the projection onto the y-z plane. The Cartesian components of the random vector, obtained using a transformation from spherical to cartesian coordinates, are

$$\hat{e}_x = \cos\theta$$

$$\hat{e}_y = \sin\theta\cos\phi$$

$$\hat{e}_z = \sin\theta\sin\phi.$$

In order to conserve momentum, the center-of-mass velocity $\vec{v}_{cm}$ must remain unchanged by the collision. The magnitude of the relative velocity is changed by an amount determined by the energy of the reaction. If $\vec{v_r}$ is the pre-collision relative velocity and ξ is the energy of reaction, the post-collision relative velocity, ${v_r}^*$, is given by

$${v_r}^{*2} = v_r^2 - 2\xi\frac{m_C + m_D}{m_C * m_D}.$$

Note that this equation has no real solutions if the quantity $v_r^2 - 2\xi\frac{m_C+m_D}{m_C*m_D}$ is negative. Physically, this is the case when the particles do not have enough kinetic energy to initiate the reaction. Such

Figure 4.10: **Collision Procedure**

1. Select the two particles for collision.
2. Compute $m_A, m_B, \vec{v_A}, \vec{v_B}$.
3. Determine the center-of-mass velocity, $\vec{v_{cm}}$.
4. Find the relative velocity and speed, $\vec{v_r}$, v_r.
5. Select a random unit vector $\hat{e}$ from a uniform distribution in space.
6. Compute the post - collision velocities.

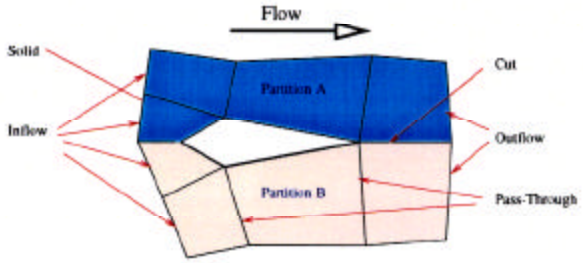

Figure 4.11: Surface Types

collisions are rejected. Once a solution for the post-collision relative velocity can be found, post-collision velocities of the two products are computed, using

$$\vec{v}_C = \vec{v}_{cm} + \frac{m_D}{m_C + m_D}\vec{v_r}$$

$$\vec{v}_D = \vec{v}_{cm} - \frac{m_C}{m_C + m_D}\vec{v_r}.$$

The procedure for computing the post-collision velocities is summarized in Figure 4.6.

4.7 Surface Types

This implementation assumes a boundary-fitted grid; except for movement and reactions, particle states are only altered at cell faces. Surface properties are used to associate information with cell faces. Each cell face has exactly one surface type that may represent a physical boundary condition (solid, inflow, outflow) or an implementation- defined property (pass-through, cut). The surface type on a cell face determines what happens to particles hitting that face, and whether

Figure 4.10: **Collision Procedure**

1. Select the two particles for collision.
2. Compute $m_A, m_B, \vec{v_A}, \vec{v_B}$.
3. Determine the center-of-mass velocity, $\vec{v_{cm}}$.
4. Find the relative velocity and speed, $\vec{v_r}$, v_r.
5. Select a random unit vector $\hat{e}$ from a uniform distribution in space.
6. Compute the post - collision velocities.

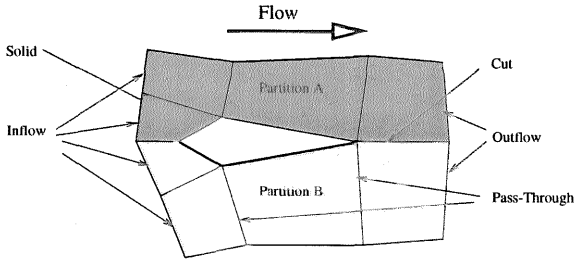

Figure 4.11: Surface Types

collisions are rejected. Once a solution for the post-collision relative velocity can be found, post-collision velocities of the two products are computed, using

$$\vec{v_C} = \vec{v_{cm}} + \frac{m_D}{m_C + m_D}\vec{v_r}$$

$$\vec{v_D} = \vec{v_{cm}} - \frac{m_C}{m_C + m_D}\vec{v_r}.$$

The procedure for computing the post-collision velocities is summarized in Figure 4.6.

4.7 Surface Types

This implementation assumes a boundary-fitted grid; except for movement and reactions, particle states are only altered at cell faces. Surface properties are used to associate information with cell faces. Each cell face has exactly one surface type that may represent a physical boundary condition (solid, inflow, outflow) or an implementation- defined property (pass-through, cut). The surface type on a cell face determines what happens to particles hitting that face, and whether

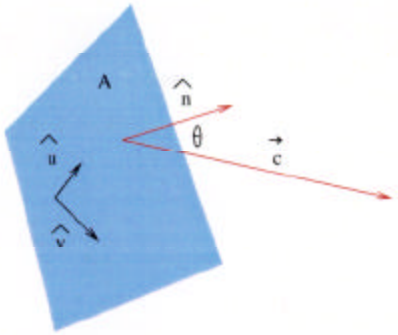

Figure 4.12: Particle Inflow

particles are injected through that face. Figure 4.11 shows two partitions of a simplified grid for flow past a solid body.

Pass-Through

The default surface type is pass-through. This type is used between adjacent cells in the same partition. No particles are injected through pass-through surfaces. Particles hitting these surfaces pass through to the adjacent cells, as described in Section 4.4. See Figure 4.11 for examples.

Inflow

Particle inflow through cell faces is the primary mechanism for introducing mass and momentum into a simulation. In Figure 4.11, for example, the upstream cell faces are designated as inflow-type surfaces. Consider a cell face with area A, unit normal $\hat{n}$, and tangential vectors $\hat{u}$ and $\hat{v}$, as shown in Figure 4.12. Assume that this face is designated to have inflow of density ρ, at temperature T, with stream velocity $\vec{c}$. In the case of multiple species, species fractions f_i will be specified. Let θ be the angle between the face normal and the stream velocity, and β_i be the temperature parameter $\beta_i = \sqrt{\frac{m_i}{2kT}}$.

Inflow Count

The molecular speed ratio s_i is given by $s_i = |\vec{c}|\beta_i$. The particle flux Φ_i can be computed, as described in [Bird94], using,

$$\Phi_i = \exp[-(s_i \cos\theta)^2 + \sqrt{\pi} s_i \cos\theta (1 + \mathrm{erf}(s_i \cos\theta))].$$

The number of simulated particles crossing the face, per unit area, per unit time, n_i, is then,

$$n_i = \frac{\Phi_i \rho}{2 F_N \beta_i \sqrt{\pi}}.$$

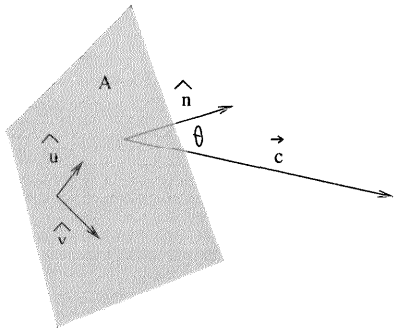

Figure 4.12: Particle Inflow

particles are injected through that face. Figure 4.11 shows two partitions of a simplified grid for flow past a solid body.

Pass-Through

The default surface type is pass-through. This type is used between adjacent cells in the same partition. No particles are injected through pass-through surfaces. Particles hitting these surfaces pass through to the adjacent cells, as described in Section 4.4. See Figure 4.11 for examples.

Inflow

Particle inflow through cell faces is the primary mechanism for introducing mass and momentum into a simulation. In Figure 4.11, for example, the upstream cell faces are designated as inflow-type surfaces. Consider a cell face with area A, unit normal $\hat{n}$, and tangential vectors $\hat{u}$ and $\hat{v}$, as shown in Figure 4.12. Assume that this face is designated to have inflow of density ρ, at temperature T, with stream velocity $\vec{c}$. In the case of multiple species, species fractions f_i will be specified. Let θ be the angle between the face normal and the stream velocity, and β_i be the temperature parameter $\beta_i = \sqrt{\frac{m_i}{2kT}}$.

Inflow Count

The molecular speed ratio s_i is given by $s_i = |\vec{c}|\beta_i$. The particle flux Φ_i can be computed, as described in [Bird94], using,

$$\Phi_i = \exp[-(s_i \cos\theta)^2 + \sqrt{\pi} s_i \cos\theta(1 + \mathrm{erf}(s_i \cos\theta))].$$

The number of simulated particles crossing the face, per unit area, per unit time, n_i, is then,

$$n_i = \frac{\Phi_i \rho}{2F_N \beta_i \sqrt{\pi}}.$$

The average inflow N_i during timestep Δt is given by,

$$N_i = n_i A \Delta t f_i.$$

This result is randomly rounded to determine the integer number to be injected during the timestep. For example, if the average is 2.9, it will be rounded to 3 with probability 0.9, otherwise rounded to 2.

Inflow Velocity

The velocity of an injected particle is determined using an acceptance-rejection sampling method, as described in [Bird94]. Let $u_n = \vec{c} \cdot \hat{n}$ be the normal component of the velocity, and let $s_n = u_n \beta$. A dimensionless speed u must be selected with probability taken from the distribution function

$$f(u) = ue^{-(u-s_n)^2}.$$

This is accomplished by computing a random velocity $u = (s_n + T)R_1$ where T is the maximum number of standard deviations to consider (for example, 4) and R_1 is a random fraction. The distribution function $f(u)$ is evaluated at this speed value. If another random fraction, R_2, is greater than $f(v)$, then u is accepted. Otherwise, a new u is computed. In other words, a value u is accepted with probability $f(u)$.

The dimensionless speed is then converted to a normal speed $v_n = \frac{u}{\beta}$, and the normal velocity is calculated as $\vec{v_n} = v_n \hat{n}$. A random thermal speed at the appropriate temperature is computed using another random number, R_3, using

$$v_{rt} = \frac{3kT}{m} \sqrt{-lnR_3}.$$

θ represents the angle that the tangential speed will make with respect to the tangential vector $\hat{v}$. This is computed as $\theta = 2\pi R_4$. The thermal components of the tangential velocity, $\vec{v_u}$ and $\vec{v_v}$ are given by

$$\vec{v_u} = v_{rt} \sin\theta \hat{u}$$

$$\vec{v_v} = v_{rt} \cos\theta \hat{v}.$$

The thermal tangential velocity, $\vec{v_{tt}}$ is the sum of these two components,

$$\vec{v_{tt}} = \vec{v_u} + \vec{v_v}.$$

The net tangential velocity $\vec{v_{nt}}$ is the difference between the mean velocity $\vec{c}$ and the normal velocity $\vec{v_n}$,

$$\vec{v_{nt}} = \vec{c} - \vec{v_n}.$$

The complete tangential velocity is the sum of the mean and thermal tangential velocities,

$$\vec{v_t} = \vec{v_{tt}} + \vec{v_{nt}}.$$

The final injection velocity $\vec{v_i}$ is then computed as the sum of normal and tangential velocities,

$$\vec{v_i} = \vec{v_t} + \vec{v_n}.$$

Once the velocity of an inflowing particle has been determined, its initial position is set to the center of the inflow face. (A better technique would be to randomly distribute inflow particles over the entire face.) After injection, inflowing particles are moved for a random fraction of the global timestep Δt, avoiding artificial clustering of the particles within a cell. Any particle hitting an inflow surface is immediately ejected from the domain.

Outflow

Outflow surfaces are equivalent to inflow-type surfaces with zero flow density. That is, no particles are injected through outflow surfaces, and particles hitting outflow surfaces are immediately ejected from the domain. This is implemented by adding them to the free list, as described in Section 4.10. In the flow example presented in Figure 4.11, the downstream faces are configured as outflow.

Cut

The parallel algorithm introduces a cut surface at each cell face located on the boundary between partitions, as shown in Figure 4.11. No particles are injected through these boundaries. A particle arriving at a cut surface is packed into a message to be sent to the appropriate neighboring partition. The time that it has taken the particle to reach the cut is subtracted from the global timestep to obtain the remaining time during which the particle must be moved upon arrival.

Solid

A solid surface is one through which particles cannot pass. These are typically used to represent rigid objects, such as metal plates or wafers. In Figure 4.11, for example, solid surfaces are used to represent the object in the center of the flow. A particle arriving at a solid surface may become embedded in the surface, may bounce elastically, or may leave the surface with a velocity determined by the temperature of the surface. The probability that a particle of a given species will become embedded in a surface is determined by an embedding probability. If a particle does not become embedded, its post-collision velocity is computed once as if the wall was fully specular (elastic), and again as if the wall was fully diffuse (accomodating). These two results are then weighted, according to a specularity coefficient, to obtain the particle's final post-collision velocity.

Thermal Accomodation

Consider a particle of a given species s, and therefore a specified mass m, at a fully-accomodating surface with temperature T. The velocity $\vec{v_a}$ of the particle upon leaving the surface must be

found. Three geometric parameters are stored for each cell face: a surface normal $\hat{n}$, and two tangential unit vectors, $\hat{u}$ and $\hat{v}$, related by the equation $\hat{u} \times \hat{v} = \hat{n}$.

The most probable velocity, v_{mp}, of a particle of mass m at temperature T, is given by

$$v_{mp} = \sqrt{\frac{2kT}{m}}.$$

The normal component of the post-collision diffuse velocity, $\vec{v_n}$, is obtained using this and a random number R_1:

$$\vec{v_n} = v_{mp}\sqrt{-\ln R_1}\hat{n}.$$

The tangential speed v_t is computed similarly, using a second random number R_2:

$$v_t = v_{mp}\sqrt{-\ln R_2}.$$

In order to distribute the velocity uniformly in the tangential direction, third random number, R_3, is used to compute the tangential angle θ_t:

$$\theta_t = 2\pi R_3.$$

The tangential direction $\hat{t}$ is computed from the tangential angle and the two tangential vectors $\hat{u}$ and $\hat{v}$,

$$\hat{t} = \hat{u}\sin\theta_t + \hat{v}\cos\theta_t.$$

The tangential velocity $\vec{v_t}$ of the particle is obtained from the tangential speed and direction using

$$\vec{v_t} = v_t\hat{t}.$$

Finally, the post-reflection velocity of the particle, assuming full accomodation, is determined by adding normal and tangential components:

$$\vec{v_a} = \vec{v_t} + \vec{v_n}.$$

Specular Reflections

Note that in the accomodation process the pre-reflection velocity is ignored completely. If the surface is not specular, however, the wall temperature is ignored and the velocity of a particle after colliding with the surface is determined exclusively by the arriving velocity of the particle $\vec{v_0}$ and the wall geometry. If the unit normal of the surface is $\hat{n}$, the component of the arriving velocity in the direction of the normal is given by $(\vec{v_0} \cdot \hat{n})\hat{n}$. The reflection process completely reverses this component so that the post-collision velocity for specular reflection $\vec{v_s}$ is given by

$$\vec{v_s} = \vec{v_0} - 2(\vec{v_0} \cdot \hat{n})\hat{n}.$$

Generalized Surfaces

Typical surfaces are partly specular and partly diffuse, and can be characterized by a specularity coefficient c_s, where $c_s = 1$ corresponds to a specular surface, and $c_s = 0$ corresponds to a diffuse surface. If a surface is neither fully accomodating nor fully specular, the specularity will be between 0 and 1. In this case, both the diffuse component $\vec{v_a}$ and the specular component $\vec{v_s}$ are computed and the final post collision velocity, $\vec{v'}$, is taken as the weighted sum of the two:

$$\vec{v'} = c_s\vec{v_s} + (1 - c_s)\vec{v_a}.$$

The ability to set the specularity coefficient provides considerable flexibility when specifying solid surface types.

Sticking

In the absence of a detailed model of surface chemistry, particle deposition can be implemented in terms of particles sticking to surfaces. That is, when a particle collides with a solid surface, it has some probability of remaining on the surface. This probability may be different for each species in the simulation. When a particle collides with solid surface, a random fraction is compared to the sticking probability for the particle's species. If the random fraction is less than the probability, the particle sticks to the surface. The particle's energy is added to the energy of the surface (for surface flux calculations) and a counter for the species and the surface is incremented (for deposition rate calculations).

Surface Emission

Solid surfaces can be configured to emit particles at the surface temperature. This is an alternative inflow method, and could be used to model sputtering or the release of etching byproducts. For an emitting surface, an emission frequency f, in particles per unit time, a surface density σ_e in number per unit area, and a fraction n_s for each species s, are specified. The sum of species fractions must be 1: $\sum_S n_s = 1$, where S is the number of species. If an emitting face has area A, the number N_s of particles of species s emitted during a timestep of length Δt is given by,

$$N_s = \frac{f\sigma_e A n_s dt}{F_N}.$$

Emitted particles are given initial velocities computed in the same method as for accomodated particles, described in Section 4.7.

4.8 Initial Conditions

The initial conditions specified for a cell include temperature, density, and species fractions of the initially injected particles. In closed problems, all particles must be introduced in this fashion. The initial conditions therefore uniquely determine the mass in the system. In steady flow problems, however, the initial conditions have no effect on the final solution. They are therefore

used to start the problem as close as possible to the correct solution, in order to minimize the time to convergence.

Given the temperature T, the total initial density ρ, and the density fractions f_i for each of the species, it is possible to determine how many of each species to inject into each cell at the beginning of the first timestep. If the cell has volume V, the number of simulated particles to inject of species i, N_i, is taken as,

$$N_i = \frac{\rho V f_i}{F_N}.$$

The velocities of initially-injected particles are based solely on the initial temperature. Given the temperature T, a random fraction R, and the particle mass m, the initial speed v_0 is computed as

$$v_0 = \frac{3kT}{m}\sqrt{-lnR}.$$

The direction of the velocity is randomly distributed in three dimensions by selecting a random unit vector $\hat{e}$, as described in Section 4.6. The initial particle velocity $\vec{v_0}$ is then,

$$\vec{v_0} = v_0\hat{e}.$$

4.9 Computing Macroscopic Parameters

Computation of macroscopic parameters such as temperature and density is performed by averaging over all of the particles in a cell. Because there are generally fewer than 10 particles per cell, significant statistical fluctuation would be present in averages taken over such small samples. For equilibrium, or time-invariant, problems, this can be countered by temporally averaging over a large number of steps. For unsteady problems, the only way to obtain smooth results is to use a large number of particles.

Mean Velocity

The mean velocity is computed for species i as,

$$\vec{\bar{v}} = \frac{1}{N_i}\sum_j \vec{v_j},$$

where the sum is over the particles in a cell of species i.

Species Temperature

The temperature of a species is computed from its velocity distribution and its mass, using

$$T_i = \frac{m_i(\bar{v_i^2} - \bar{v_i}^2)}{3k}.$$

Species Density

The density of a species is obtained by multiplying the number of simulated particles in the cell by F_N, and dividing the result by the cell volume,

$$\rho_i = \frac{F_N N_i}{V},$$

where N_i is the number of simulated molecules and V is the cell volume.

Knudsen Number

The Knudsen number for species i, Kn_i, is computed as the ratio of mean free path λ_i to characteristic length L. The characteristic length of a cell, is taken as the cube root of the cell volume. (For orthogonal cells, this is equivalent to the geometric mean of the cell side lengths.) The computation of mean free path involves evaluations of cross sectional area. These evaluations are performed using the RMS thermal velocity, $v_{t_{rms}}$, where the thermal velocity of a particle is the difference between the particle's velocity and the species mean velocity,

$$v_t = v - \bar{v}.$$

The mean free path of species i, λ_i, is computed using,

$$\frac{1}{\lambda_i} = \sum_{j=1}^{K} \rho_j \sigma_{ij},$$

where K is the number of species in the simulation, ρ_j is the density of species j, σ_{ij} is the cross section for collisions between species i and species j. Since there may be several reactions between species i and species j, it is necessary to sum over all of these to obtain the total effective cross section between two species, σ_{ij},

$$\sigma_{ij} = \sum_{r=1}^{R} \sigma_{ijr}(\bar{v}_t),$$

where R is the number of reactions between species i and j, and σ_{ijr} is the cross section for the r^{th} reaction between species i and j. The mean free path λ_i is therefore computed using,

$$\frac{1}{\lambda_i} = \sum_{j=1}^{K} \rho_j \sum_{r=1}^{R} \sigma_{ijr}(\bar{v}_t).$$

Once the mean free path and the cell length, have been determined, species Knudsen number for the cell is computed using,

$$Kn_i = \frac{\lambda_i}{L}.$$

4.10 Software Engineering

Previous DSMC implementations have used Fortran-style array-based data types. For example, they may have separate arrays for every particle attribute, indexed by particle number. Instead of a particle being directly associated with the cell that contains it, this information must be stored in a separate indexing array. This approach is undesirable because it is neither modular nor maintainable. Since the structures defining particles, and the routines that manipulate them, are scattered throughout the code, a small change in the definition of a particle requires changes throughout the code.

In order for *Hawk* to be modular and maintainable, it encapsulates the definitions of a structure and the operations on that structure, in an *Abstract Data Type* (ADT). For example, all of the characteristics of a particle are encapsulated in one definition, and the procedures that operate on particles are logically grouped together. If it is necessary to modify the definition of a particle, only these procedures must be modified. The following sections describe some of the most important ADT's in this implementation.

Partition

The partition ADT, as shown in Figure 4.13, stores a portion of the grid and a list of particles to be sent to neighboring partitions. It also maintains aggregate properties of the cells in the grid, such as the total number of particles and the average temperature. The operations on a partition include sending transitioning particles to neighboring partitions, executing timesteps on all cells in the partition, and packing the entire partition into a message for load-balancing purposes.

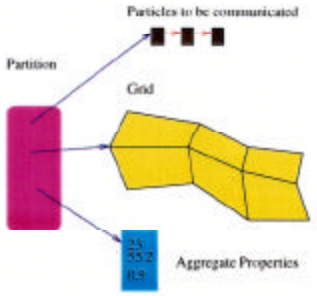

Figure 4.13: Partition ADT

Grid

As shown in Figure 4.14, a grid is represented as a collection of cells, connected through shared faces. Operations on a grid include performing a timestep on all cells, inserting a particle arriving from a neighboring partition, and loading the grid from a file.

4.10 Software Engineering

Previous DSMC implementations have used Fortran-style array-based data types. For example, they may have separate arrays for every particle attribute, indexed by particle number. Instead of a particle being directly associated with the cell that contains it, this information must be stored in a separate indexing array. This approach is undesirable because it is neither modular nor maintainable. Since the structures defining particles, and the routines that manipulate them, are scattered throughout the code, a small change in the definition of a particle requires changes throughout the code.

In order for *Hawk* to be modular and maintainable, it encapsulates the definitions of a structure and the operations on that structure, in an *Abstract Data Type* (ADT). For example, all of the characteristics of a particle are encapsulated in one definition, and the procedures that operate on particles are logically grouped together. If it is necessary to modify the definition of a particle, only these procedures must be modified. The following sections describe some of the most important ADT's in this implementation.

Partition

The partition ADT, as shown in Figure 4.13, stores a portion of the grid and a list of particles to be sent to neighboring partitions. It also maintains aggregate properties of the cells in the grid, such as the total number of particles and the average temperature. The operations on a partition include sending transitioning particles to neighboring partitions, executing timesteps on all cells in the partition, and packing the entire partition into a message for load-balancing purposes.

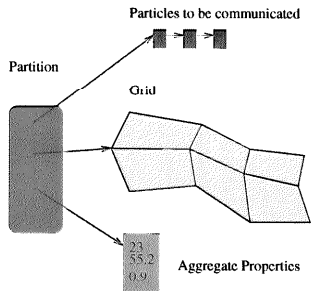

Figure 4.13: Partition ADT

Grid

As shown in Figure 4.14, a grid is represented as a collection of cells, connected through shared faces. Operations on a grid include performing a timestep on all cells, inserting a particle arriving from a neighboring partition, and loading the grid from a file.

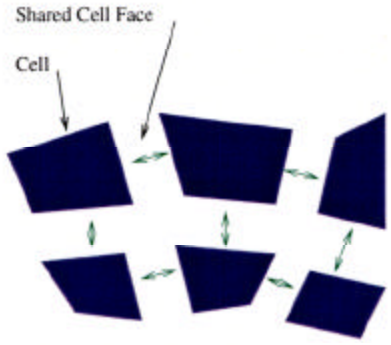

Figure 4.14: Grid ADT

Cell

Each cell maintains a list of its faces, a list of particles that it contains, and macroscopic parameters. Cell operations include moving and colliding all particles, sending a particle to a neighboring cell, and computing macroscopic parameters. See Figure 4.15.

Cell Face

Cell faces have associated boundary condition types, and energy fluxes. Operations on cell faces include particle inflow, particle reflection, and particle sticking.

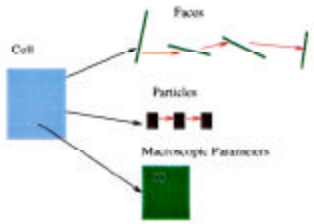

Figure 4.15: Cell ADT

Particles

The most fundamental ADT is that of a particle, which is comprised of position and velocity vectors in three dimensions and a species identifier that can be used to determine mass and charge. Particles are always stored in a circularly-linked list, as shown in Figure 4.16. This allows all particles in the cell to be visited using an efficient list traversal. It also facilitates the dynamic creation and deletion of particles, and movement of particles between cell lists. All operations on particle lists, including concatenation, are performed in constant time. Traversal of particle lists is achieved in time proportional to the length of the list.

In order to minimize memory allocation overhead and fragmentation, a list of "free" particles is maintained. Whenever a particle is destroyed (removed from the simulation), it is added to

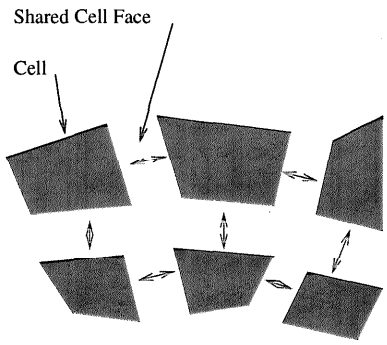

Figure 4.14: Grid ADT

Cell

Each cell maintains a list of its faces, a list of particles that it contains, and macroscopic parameters. Cell operations include moving and colliding all particles, sending a particle to a neighboring cell, and computing macroscopic parameters. See Figure 4.15.

Cell Face

Cell faces have associated boundary condition types, and energy fluxes. Operations on cell faces include particle inflow, particle reflection, and particle sticking.

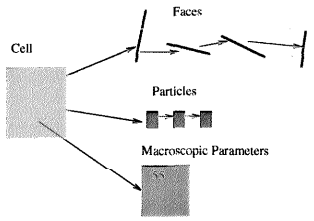

Figure 4.15: Cell ADT

Particles

The most fundamental ADT is that of a particle, which is comprised of position and velocity vectors in three dimensions and a species identifier that can be used to determine mass and charge. Particles are always stored in a circularly-linked list, as shown in Figure 4.16. This allows all particles in the cell to be visited using an efficient list traversal. It also facilitates the dynamic creation and deletion of particles, and movement of particles between cell lists. All operations on particle lists, including concatenation, are performed in constant time. Traversal of particle lists is achieved in time proportional to the length of the list.

In order to minimize memory allocation overhead and fragmentation, a list of "free" particles is maintained. Whenever a particle is destroyed (removed from the simulation), it is added to

Figure 4.16: Particle List ADT

the free list. Whenever a particle is created, if there is a particle available on the free list, the created particle is taken from the free list. If no particles are available on the free list, space for a new particle is dynamically allocated. In other words, particles are only allocated using dynamic memory allocation if no free particles exist within a computer. The free list is shared among all of the partitions in a computer, increasing the probability that free particles can be found when needed.

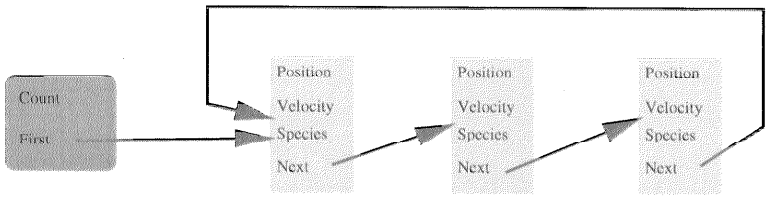

Figure 4.16: Particle List ADT

the free list. Whenever a particle is created, if there is a particle available on the free list, the created particle is taken from the free list. If no particles are available on the free list, space for a new particle is dynamically allocated. In other words, particles are only allocated using dynamic memory allocation if no free particles exist within a computer. The free list is shared among all of the partitions in a computer, increasing the probability that free particles can be found when needed.

Chapter 5

Validation

This chapter describes a sequence of validation experiments intended to verify the basic neutral flow simulation characteristics of this implementation. Simulation results are compared with theoretical predictions and experimental data. Low-level routines, such as the random number generator and the particle transport mechanism, are evaluated. A series of uniform box tests are performed to evaluate collisions, initial conditions, and sampling. The validation suite concludes with a series of tests for heat transfer and flow problems.

5.1 Random Number Generation

At the heart of any Monte Carlo code is the random number generator. DSMC codes are potentially sensitive to several possible flaws in random number generators. Common problems include short periods, non-uniformity, and correlations between consecutive numbers. In large simulations, any of these flaws can lead to erroneous results.

Numerical Recipies in C [Press92] presents a suite of random number generators of varying quality and speed. These generators were compared using a C implementation of Bird's program RANDIST [Bird94]. This test is designed to evaluate the uniformity of the generated numbers as follows. A large two 100×100 integer array is initialized to zero. One cell is selected from the array using random numbers to pick a row and a column, and the integer in that cell is incremented. 10^8 cells are selected this way, uniformly filling the array. The expected number in each cell is $10^8/10^4 = 10^4$, and the expected standard deviation is $\sigma = \sqrt{10^4} = 100$. A histogram for values outside 1, 2, 3, 4, and 5 standard deviations was computed and compared with a Gaussian distribution. If the random numbers were not uniform, the histogram would not match a Gaussian.

Table 5.1 shows the results of the study, including the histogram values, period, and execution time for the test program on a 200 MHz Silicon Graphics Indigo 2. `ran0` is the "minimal" generator of Park and Miller. `ran1` includes the Bays-Durham shuffle and added safeguards. `ran2` is a L'Ecuyer's generator with Bays-Durham shuffle and added safeguards. `ran3` is Knuth's version of a subtractive method, and `ran4` is based on pseudo - DES encryption [Press92]. Small simulations were performed using each of the generators described above and the results were identical. For large simulations, however, `ran2` was selected because of its

Table 5.1: Random Number Generator Comparison

Method	σ	2σ	3σ	4σ	5σ	time	period
theoretical	3173	455	27.0	0.63	0.0057		
`ran0`	3067	411	23	0	0	5:21.46	10^9
`ran1`	3203	484	24	2	0	4:35.13	10^9
`ran2`	3128	439	20	0	0	7:52.50	10^{18}
`ran3`	3188	449	29	0	0	2:42.98	
`ran4`	3157	442	25	1	0	11:18.49	

long period ($> 2 \times 10^{18}$). More extensive verification would include full three-dimensional uniformity tests and checking of periods. Since `ran2` is among the most advanced generators available, however, it should be more than adequate for the purposes of this project.

5.2 Isothermal Box Tests

Having confirmed the correctness of the random number generator the next series of tests was concerned with basic operations, such as particle transport and collisions. These tests were performed in a simple box grid that was generated using ICEM-CFD/MULCAD. Initial and boundary conditions were varied to simulate a variety of physical situations.

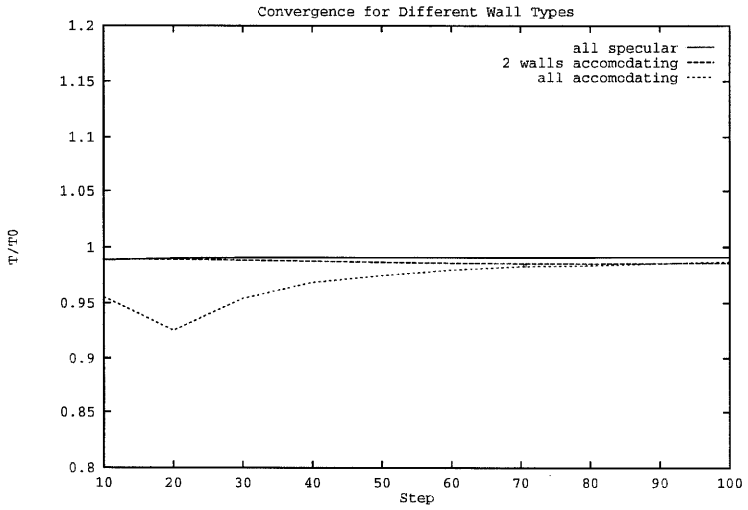

Figure 5.1: Convergence for Different Wall Types

Most simulations begin with the placement of a number of particles in the domain, using a specified temperature and density. In order to verify that this procedure was correctly implemented, the box grid was filled with particles in this manner and the macroscopic parameters were sampled immediately thereafter. The sampled values were observed to be within statistical scatter of the specified values.

In a box with specular walls, no amount of particle movement should change the macroscopic paramcters. A series of tests in a fully-specular box verified this. Macroscopic parameters remained close to, and converged towards, the specified initial conditions. No change in parameters was observed across the domain, suggesting that particle transport and specular reflection are correctly implemented.

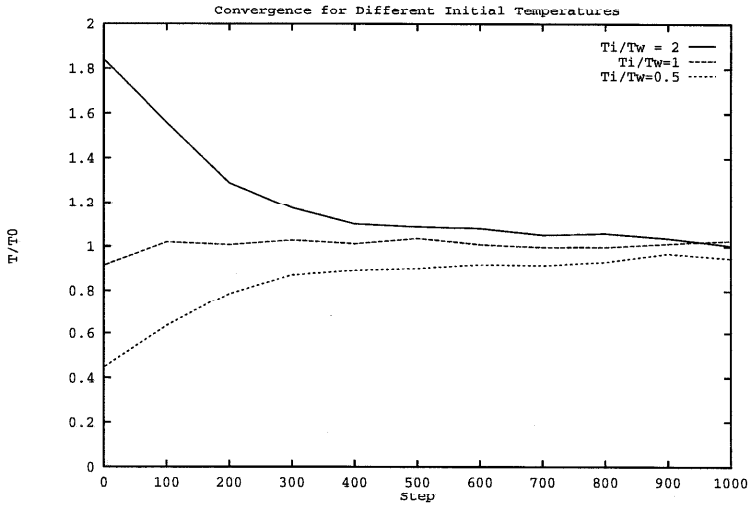

Figure 5.2: Convergence for Different Initial Temperatures

Diffuse surface accomodation, the process by which an arriving particle equilibrates to the wall temperature, is important in heat transfer problems. A test for this procedure was a box grid with some walls accomodating and the rest specular. If all diffuse walls are set to the same temperature T_w, the temperature everywhere in the domain should converge to that temperature, regardless of the initial temperatre, and regardless of how many of the walls are accomodating. Figure 5.1 shows the convergence history for different numbers of accomodating walls. It is clear that the cell temperatures converge to the wall temperature.

A further test of the accomodating routines is to show that the internal temperature converges to the wall temperature regardless of the initial temperatre. Figure 5.2 shows the convergence history for different initial temperatures. It is clear from this study that the initial conditions affect the rate of convergence of the solution, but not its correctness.

The preceding isothermal box tests show that the basic operations of particle injection, particle transport, specular and accomodating surfaces, and macroscopic parameter computation are correct. Given the proper functioning of these low-level operations, the collision process can be studied in detail.

5.3 Collisions

Collisions between particles are central to the DSMC technique because they allow for macroscopic gradients in flow fields. *Hawk*'s collision routines were studied in terms of collision fre-

quency and elasticity. As the collision frequency is a function of relative velocity, the first test was to verify that the mean relative velocity between particles is equal to that predicted by kinetic theory. For a gas at equilibrium, the mean realtive velocity $\overline{v_{rel}}$ between particles of mass m at temperature T is given by,

$$\overline{v_{rel}} = \sqrt{\frac{16kT}{\pi m}}.$$

Correct calculation of the number of collision tests was verified by printing this number and comparing it to the expected number,

$$N_{max} = \frac{N^2[v_{rel}\sigma(v_{rel})]_{max}}{2V},$$

where N is the number of particles in the cell, v_{rel} is the relative velocity, σ is the collision cross section, and V is the cell volume. The acceptance-rejection collision technique is designed to reproduce the correct number of actual collisions, given by,

$$N_{coll} = \frac{N^2\overline{v_{rel}\sigma(v_{rel})dt}}{2V},$$

where dt is the timestep and V is the cell volume. For validation purposes, the average number of collisions was computed for each cell and compared to the expected number obtained from this equation. The agreement was excellent.

The elasticity of the collisions was verified by computing the total translational energy in the system. Simulations of a box with specular walls and colliding particles showed no change in total energy even after thousands of timesteps. For additional verification, pre- and post- collision energies were computed and shown to be equal.

5.4 Velocity Distribution Functions

Velocity distribution functions are important macroscopic parameters. A good check on *Hawk*'s operation was to compute the velocity distribution functions in different cells and different directions. These functions, for a uniform box test, were shown to be identical to the theoretically predicted Maxwellian distribution. The Maxwellian distribution function is given by,

$$f(v) = e^{\frac{-m(v-\bar{v})^2}{2kT}}dv,$$

where m is the particle mass, $\bar{v}$ is the net velocity, T is the cell temperature, and k is the Boltzmann constant. Separate histograms and Maxwellians were computed in each cell for each direction.

Figure 5.3 shows velocity distribution functions in the x direction for three different cells in the domain, as well as the theoretically-predicted Maxwellian distribution. Note the excellent agreement.

Figure 5.4 shows the distribution functions in the x, y, and z directions. The function in each direction is identical to the theoretical function. That there is no difference between the three directions suggests that the code is valid in three dimensions.

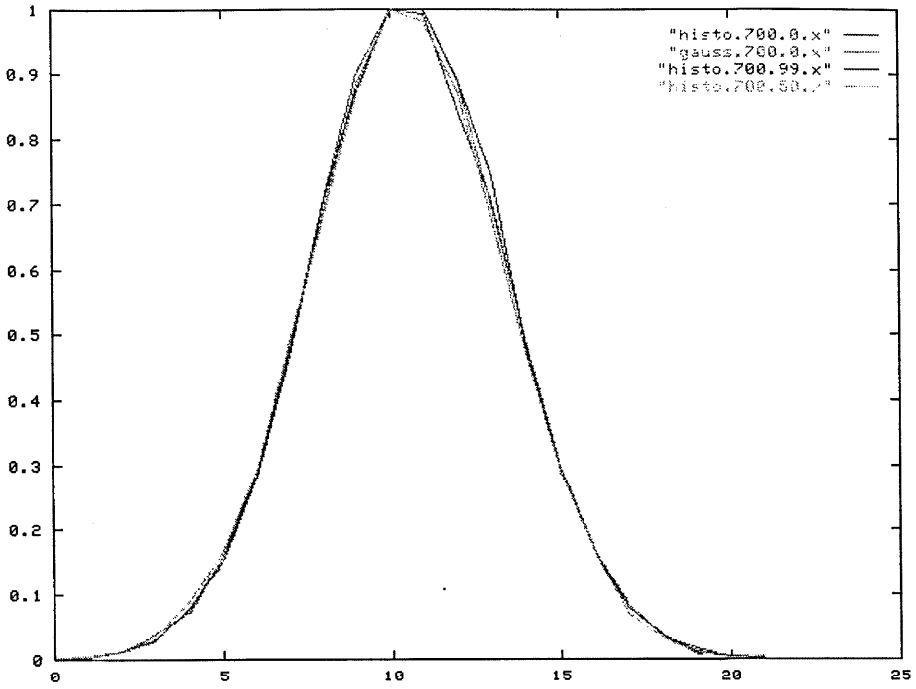

Figure 5.3: Distribution functions in x direction, for different cells

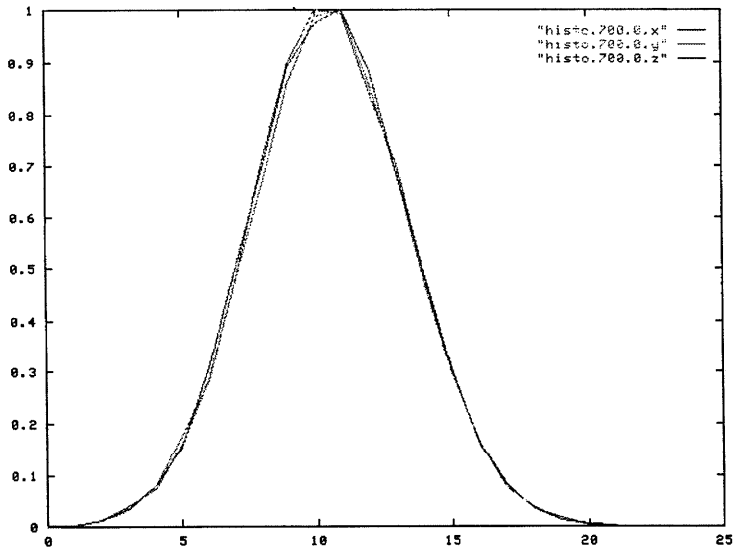

Figure 5.4: Distribution functions in x, y, and z directions

5.5 Heat Transfer

The standard heat transfer problem is one-dimensional in nature, described by two walls at temperatures T_1 and T_2, separated by a distance L. Macroscopic parameters are computed at positions y between the walls. If these parameters are plotted as a function of the normalized position, $\frac{y}{L}$, the problem can be completely specified by the temperature ratio $\beta = \frac{T_1}{T_2}$ and the Knudsen number of the system.

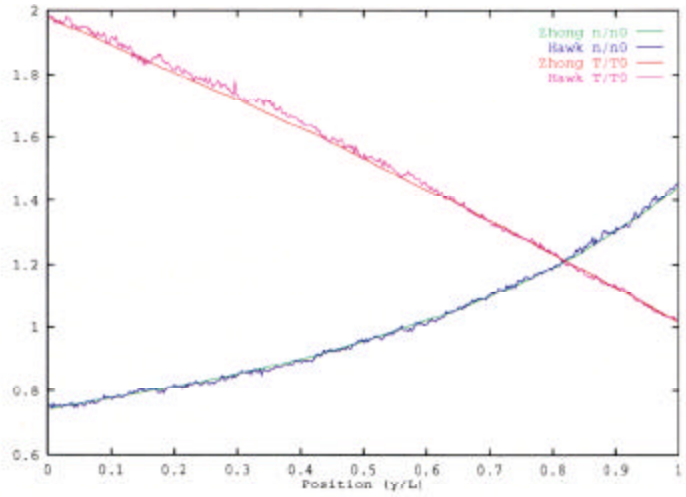

Figure 5.5: $Kn = 0.01$, $\beta = 2$

Simulation results were quantitatively validated by comparison against another DSMC code. A convenient example was the study of the heat transfer problem presented by Zhong and Koura [Zhong95]. These cases are for one-dimensional problems in which both the Knudsen number and β are varied. Simulations were performed using a 400-cell grid and an average of 10 particles per cell. Execution times on one processor of an SGI Power Challenge were 4-6 hours.

Figure 5.5, for $Kn = 0.01$ and $\beta = 2$ shows a curved density profile and a linear temperature profile that matches the wall temperatures exactly. The *Hawk* results match Zhong's results perfectly. Figure 5.6, for Kn = 0.01 and $\beta = 20$, shows a sharp increase in density near the cold wall. Notice that the temperature profile is no longer linear, and stops short of the wall temperatures.

For Kn = 0.1, there are fewer collisions and the system is unable to hold as steep a temperature gradient, as shown in Figure 5.7. The temperature by the hot wall is therefore 10-15% below the wall temperature, and the temperature by the cold wall is 10% above the cold wall temperature.

For $\beta = 20$, in Figure 5.8, there is a linear temperature profile, except for curves towards the wall temperatures in cells near the walls.

Comparisons with Zhong's data show excellent agreement. The remaining tests in Zhong's paper involve moving plates and periodic boundary conditions. Since these are not important to plasma or satellite simulations, they have not been included in this series of validation tests. From the tests presented here, it is clear that *Hawk* 's results for the heat transfer problem are quantitatively correct.

5.5 Heat Transfer

The standard heat transfer problem is one-dimensional in nature, described by two walls at temperatures T_1 and T_2, separated by a distance L. Macroscopic parameters are computed at positions y between the walls. If these parameters are plotted as a function of the normalized position, $\frac{y}{L}$, the problem can be completely specified by the temperature ratio $\beta = \frac{T_1}{T_2}$ and the Knudsen number of the system.

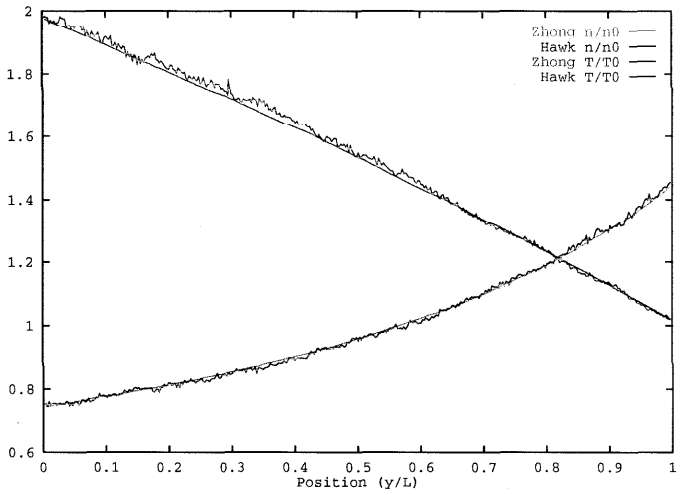

Figure 5.5: $Kn = 0.01$, $\beta = 2$

Simulation results were quantitatively validated by comparison against another DSMC code. A convenient example was the study of the heat transfer problem presented by Zhong and Koura [Zhong95]. These cases are for one-dimensional problems in which both the Knudsen number and β are varied. Simulations were performed using a 400-cell grid and an average of 10 particles per cell. Execution times on one processor of an SGI Power Challenge were 4-6 hours.

Figure 5.5, for $Kn = 0.01$ and $\beta = 2$ shows a curved density profile and a linear temperature profile that matches the wall temperatures exactly. The *Hawk* results match Zhong's results perfectly. Figure 5.6, for Kn = 0.01 and $\beta = 20$, shows a sharp increase in density near the cold wall. Notice that the temperature profile is no longer linear, and stops short of the wall temperatures.

For Kn = 0.1, there are fewer collisions and the system is unable to hold as steep a temperature gradient, as shown in Figure 5.7. The temperature by the hot wall is therefore 10-15% below the wall temperature, and the temperature by the cold wall is 10% above the cold wall temperature.

For $\beta = 20$, in Figure 5.8, there is a linear temperature profile, except for curves towards the wall temperatures in cells near the walls.

Comparisons with Zhong's data show excellent agreement. The remaining tests in Zhong's paper involve moving plates and periodic boundary conditions. Since these are not important to plasma or satellite simulations, they have not been included in this series of validation tests. From the tests presented here, it is clear that *Hawk* 's results for the heat transfer problem are quantitatively correct.

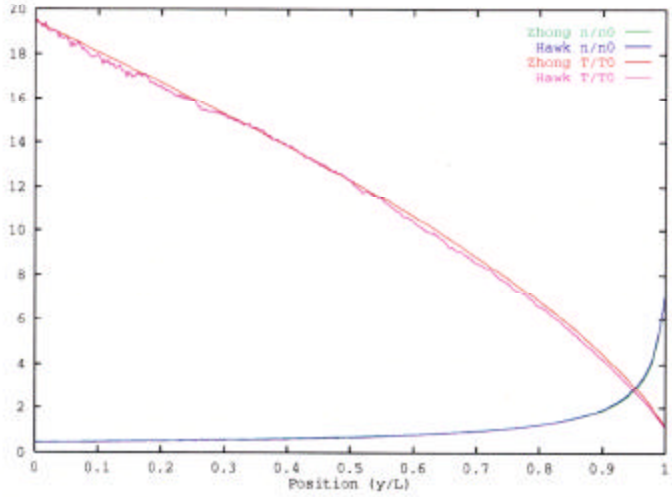

Figure 5.6: $Kn = 0.01, \beta = 20$

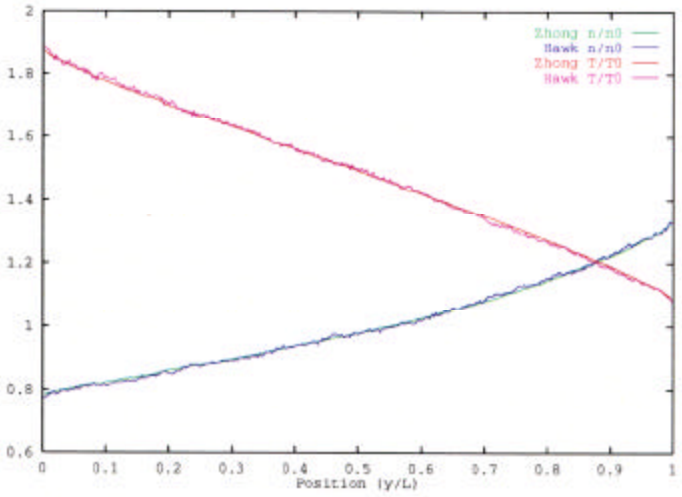

Figure 5.7: $Kn = 0.1, \beta = 2$

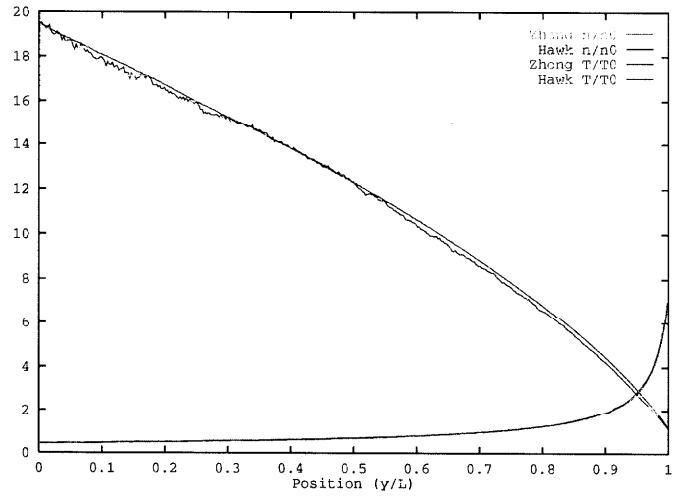

Figure 5.6: $Kn = 0.01$, $\beta = 20$

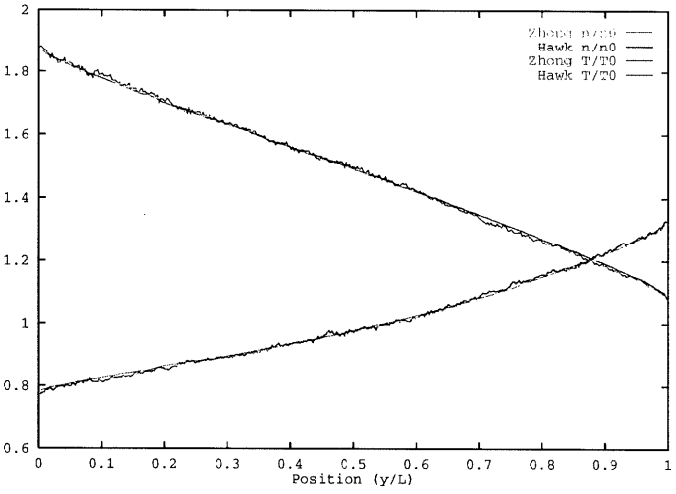

Figure 5.7: $Kn = 0.1$, $\beta = 2$

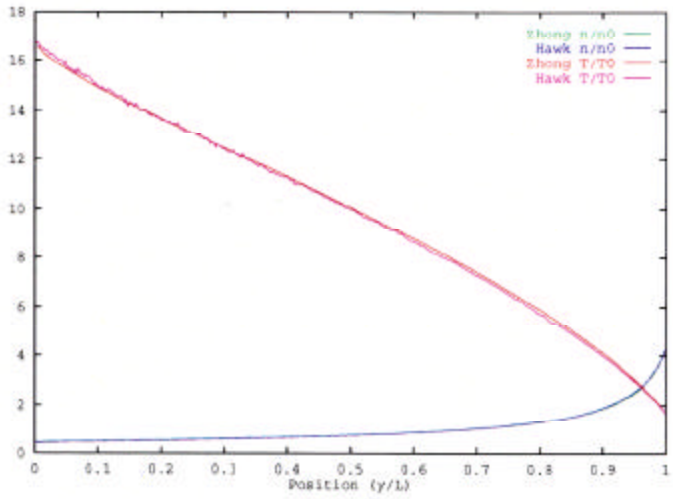

Figure 5.8: $Kn = 0.1, \beta = 20$

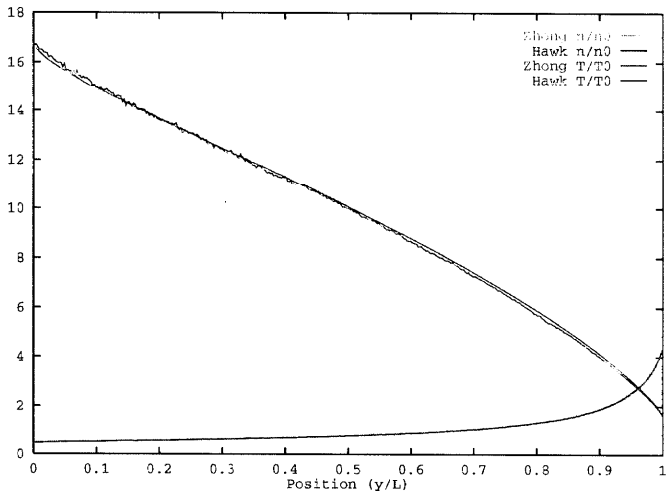

Figure 5.8: $Kn = 0.1$, $\beta = 20$

5.6 Sensitivity to the Number of Particles

The cost of the DSMC method, both in terms of run time and storage requirements, is proportional to the number of simulated particles used. It is therefore important to know the minimum number of particles necessary to reproduce accurate statistics. This was determined by simulating a heat transfer problem, varying the number of simulated particles per cell. In order to achieve comparable statistical properties, the product of particle count and timesteps was held constant. Figure 5.9 shows that the results were identical for 1, 10, or 100 particles per cell, but that with only 0.1 particles per cell there was a significant loss in accuracy. Even with only 0.1 particles per cell, though, qualitative aspects of the flow were still captured.

The quality of the simulations for small numbers of particles can be attributed to the exponential distribution of possible collisions [Ivanov88, Ivanov88a, Ivanov91a]. Without the use of that approach, the collision routine would have been unable to capture any of the macroscopic trends for small numbers of particles.

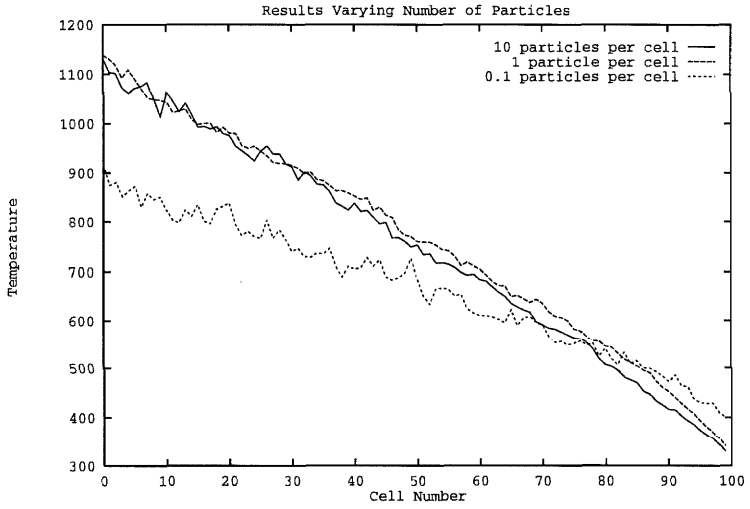

Figure 5.9: Comparison of Number of Particles

5.7 Flow Tests

Most interesting plasma problems involve an inflow of particles into the domain. In satellite simulations, this is in the form of high mach flow impingent upon the front of the satellite. In reactors, particles arrive through low-velocity pumping ports. It is therefore important for *Hawk* to include a general inflow mechanism that is valid for both high and low mach numbers. For each inflow cell face, the normal and tangential velocity components are sampled from separate distribution functions [Bird94].

These routines were tested by running flow through a tube in a variety of configurations. The average cell densities and net velocities were then compared with those specified for the inflow

and found to be in agreement.

5.8 Parallel Algorithm

As *Hawk* is designed to simulate large problems on parallel computers, it is important to verify that simulations run in parallel give the same results as those run sequentially. In *Hawk* 's parallel implementation, particles are transported between partitions of the grid through messages between processors. In order to validate the parallel algorithm, the same problem was simulated on different numbers of processors and the results were compared. Figure 5.10 shows the results from three of the configurations. Note that they are identical to within statistical scatter. The parallel implementation therefore preserves the correct physics of the sequential version.

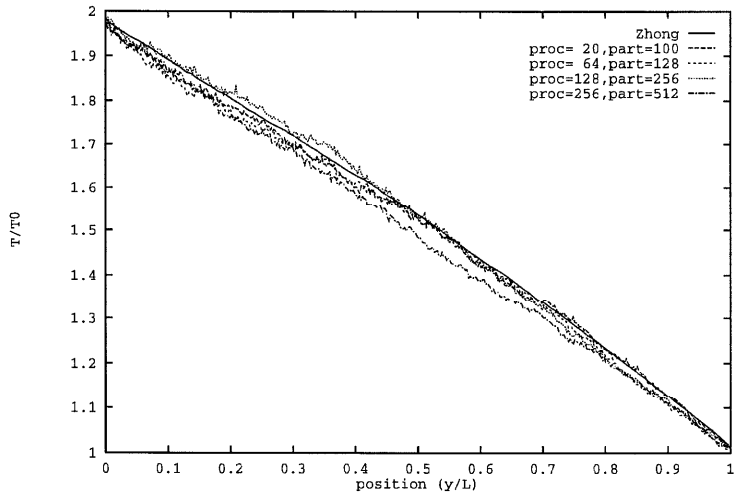

Figure 5.10: Parallel Results

Chapter 6

Large-Scale Simulations

6.1 The GEC Reference Cell

The Gaseous Electronics Conference (GEC) reference cell is a standard plasma reactor used in a variety of applications. The first simulation of any plasma reactor should consider neutral flow in a representative configuration. These results should be compared with experimental data to validate the model and the operating conditions. Unfortunately, no experimental data is available for realistic three-dimensional configurations of the GEC cell. While the authors are collaborating with experimentalists to obtain data for comparison, only simulation results can be presented at this time.

Figure 6.1 shows the outside of the GEC Reference cell, with five of the eight ports visible. Figure 6.2 shows the inside of the reactor. The wafer is the disc sitting on the bottom pedistal. Induction coils are located inside the upper and lower cylinders.

Simulations were configured as follows. All walls were fully accomodating at 300K. Inflow was through a small tube with a temperature of 300K, density of 10^{19} particles/m^3, and velocity 188 m/s. Initial conditions were a uniform distribution of particles at a temperature of 300K and density 5.17×10^{19} particles/m^3, and with no net velocity. Neutral chlorine atoms (Cl_2) were used, with mass 7.1×10^{-26} kg and cross section 1e-19 m^2.

Simulations of the GEC reference cell were performed using both a 512-node Intel Paragon and a 4-processor SGI Power Challenge. Simulation times were approximately 24 hours on the Paragon, and 300 hours on the SGI. Memory requirements were approximately 300 Mb for a 500,000 particle simulation, and 1.5 Gb for a 2.8 million particle simulation. No attempt has yet been made to optimize simulation execution time.

Simulation Grid

The grid for this simulation was generated from engineering drawings using ICEM-CFD. It is a $95 \times 93 \times 66$ structured hexahedral grid, with 580,000 grid cells, of which approximately 330,000 are active. The remaining cells, are located outside of the reactor and contain no particles.

Figure 6.3 shows a perspective view of the external surfaces of the GEC grid. For operation at low pressures, the reactor is electromagnetically driven by coils above and below the wafer.

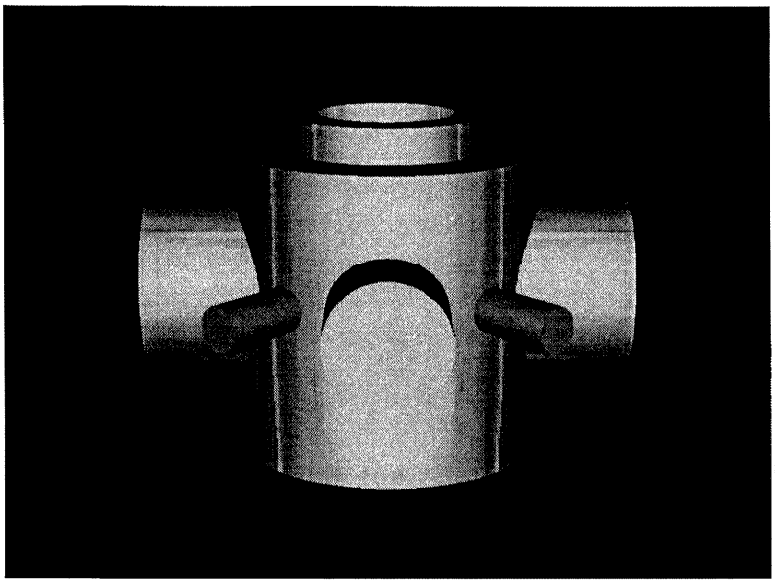

Figure 6.1: Outside of the GEC Reference Cell

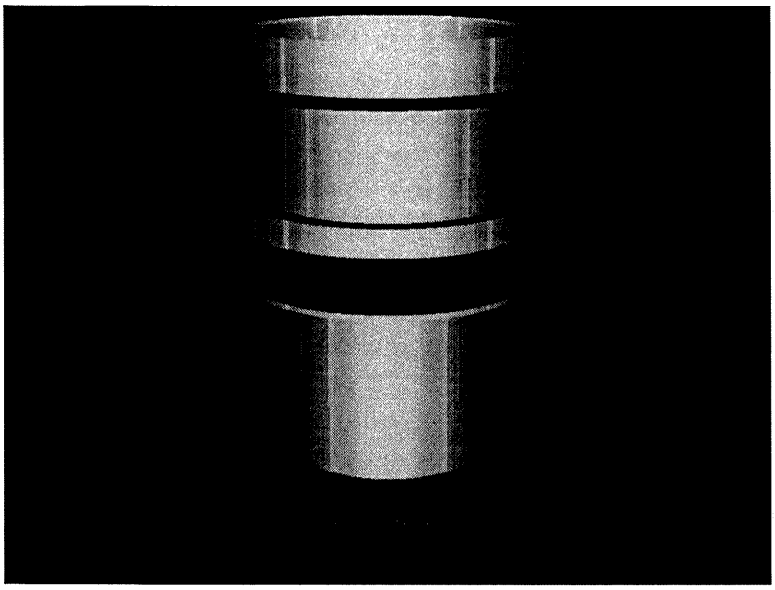

Figure 6.2: Inside of the GEC Reference Cell

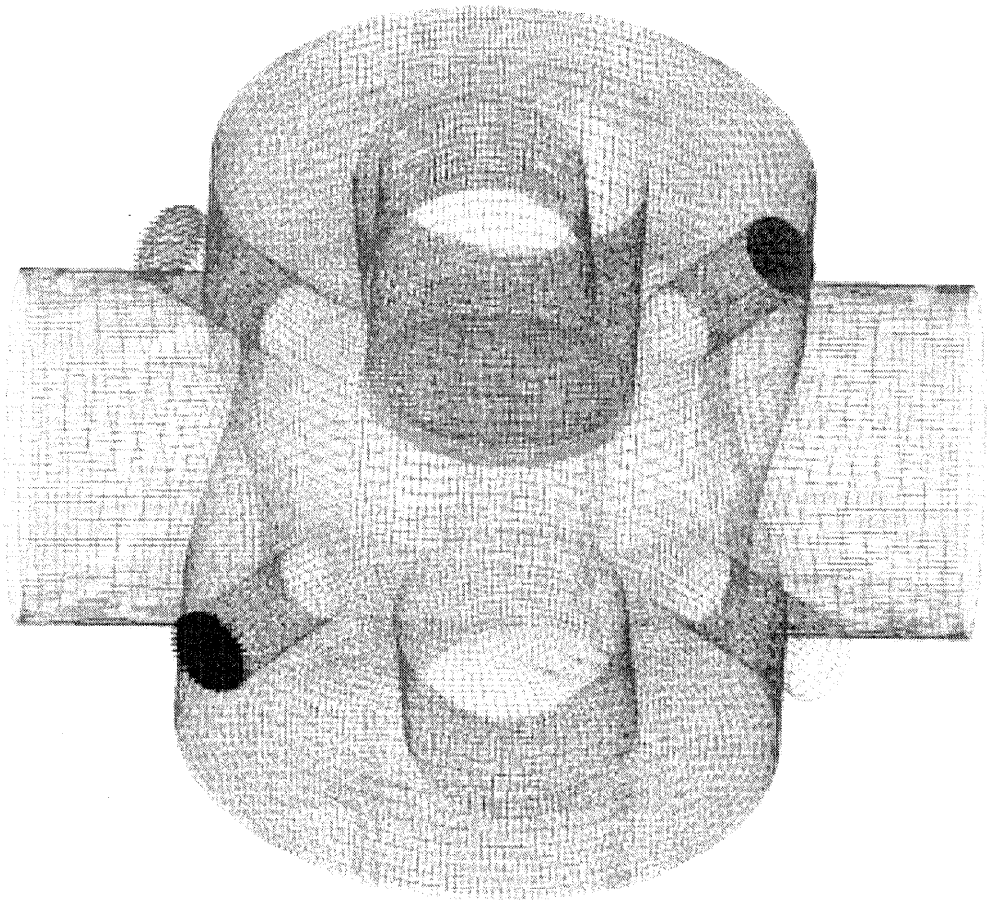

Figure 6.3: The GEC grid

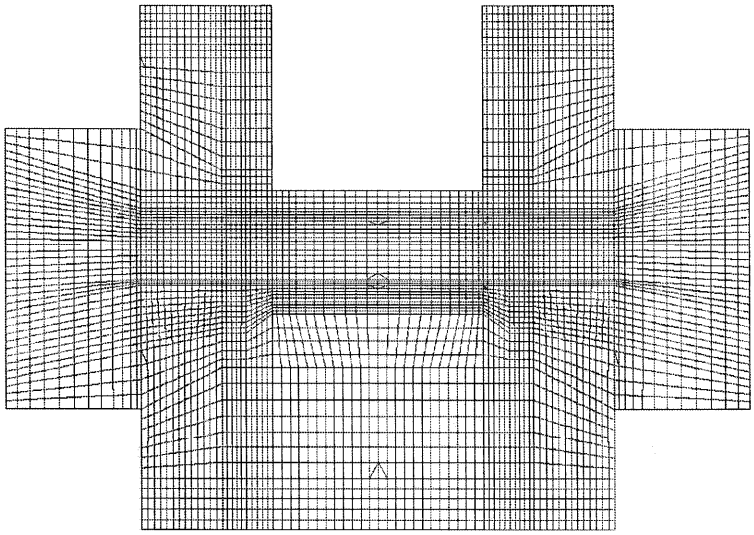

Figure 6.4: Vertical Slice through the GEC Grid

For this reason, injection and exhaust must be through the tubes at the sides. In this case, exhaust is through the large port on the left, inflow is through the small port in the right foreground. The location of these ports gives rise to interesting asymmetric effects that would be impossible to reproduce with an axisymmetric code. It is also clear that such a complex grid could not be generated by hand. Support for automatically generated grids is essential.

Figure 6.4 shows a vertical slice of the GEC grid, taken through the large ports. The wafer is described by the group of thin cells just below the center of the grid. The central region below the wafer has grid cells, but is isolated from the flow. Figure 6.5 shows a horizontal slice through the middle of the reactor. The inflow port is on the bottom right, and the outflow port is on the left.

Simulation Overview

Two simulations of the GEC reference cell were performed, in order to demonstrate the scalability and portability of the code, and to study convergence properties. The first was performed on 512 processors, each with 32 Mb RAM, on an Intel Paragon at Caltech. The second test case was simulated on 4 processors of an SGI Power Challenge with 1 Gb of shared memory. The appropriate timestep for these calculations is not yet known. These experiments are part of an ongoing sensitivity analysis test. Results of the two cases were compared in terms of number of timesteps to convergence, smoothness, and macroscopic flow properties.

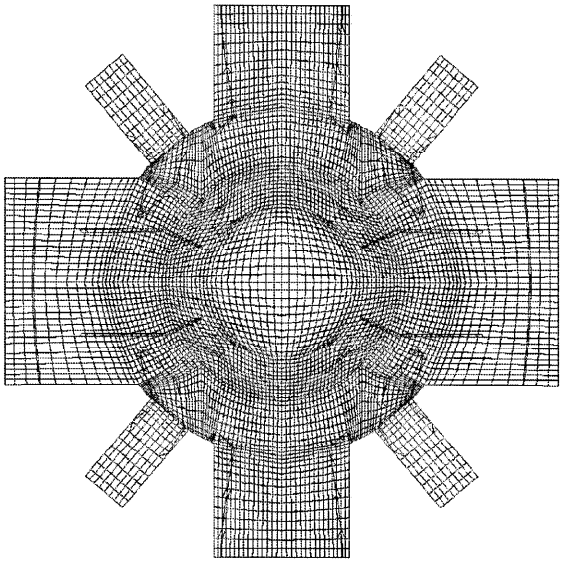

Figure 6.5: Horizontal Slice through the GEC Grid

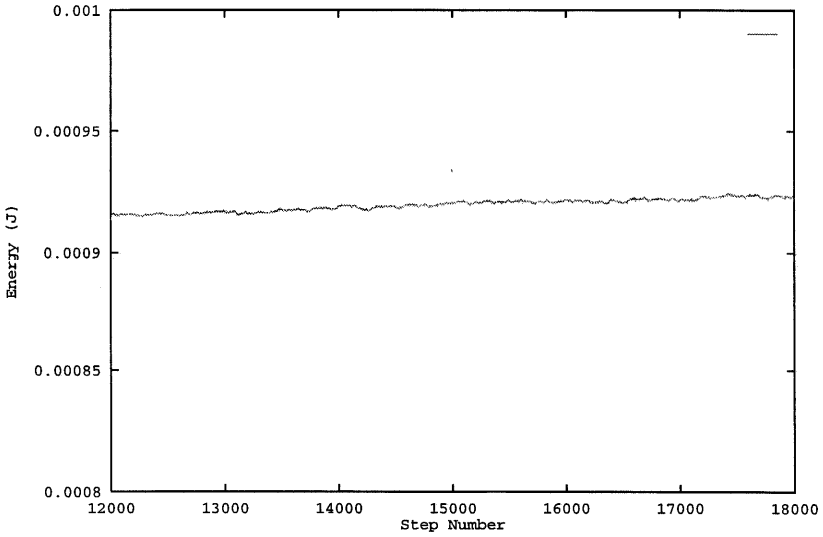

Figure 6.6: System Kinetic Energy

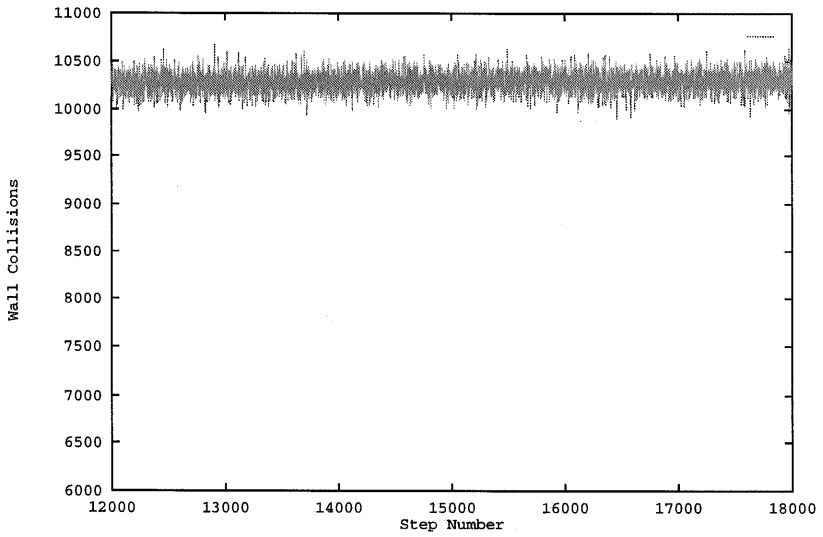

Figure 6.7: Wall Collisions

6.2 Case I

In this case, each simulated particle represented 10^{11} real particles, yielding a total of 1.4 million simulated particles. The global timestep was 5×10^{-7} seconds, a value selected so that a typical particle requires at least 2-3 timesteps to traverse a cell of average size.

Convergence History

Determining the convergence for DSMC simulations is not straightforward. One approach is to observe changes in macroscopic properties of the system as a whole as a function of timestep number. A plot of the number of particles in the system as a function of timestep number, Figure 6.8, clearly demonstrates that this simulation has not converged. The number of particles is increasing, almost linearly, in the range of the plot.

Similarly, Figure 6.6 shows that the energy is still increasing in the system, though less rapidly. The increase in energy is most likely a direct result of the increase in particles. Figure 6.7, on the other hand, shows the number of wall collisions in the system as a function of timestep, and this appears to be constant. Wall collisions, therefore, are a less sensitive measurement.

Macroscopic Parameters

Figure 6.9 shows the density in a horizontal plane just above the wafer. Note the high-density inflow through the small tube at the lower right, and density decreasing in the outflow pipe on the left. More importanly, notice that density above the wafer is higher near the inflow port than near the outflow port. This could induce a non-uniformity in the etching rate throughout the wafer, yielding poor quality wafers.

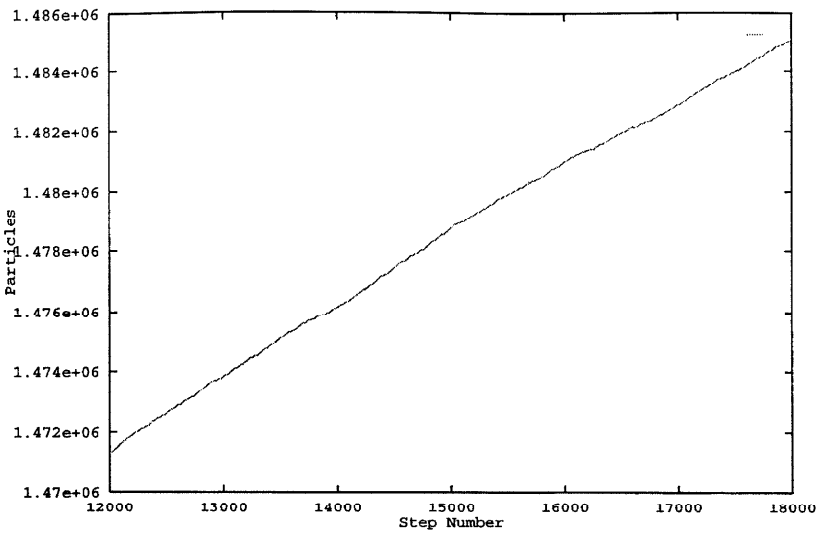

Figure 6.8: Particle Count

Figure 6.10 shows the average velocity of gas particles within each cell (with colors proportional to particle speed). The arrows clearly show an expansion of the gas upon entering the central region, as well as circulation zones on either side of the inlet junction. Over the wafer, the gas turns from the inflow angle to the outflow angle, introducing a noticeable asymmetry.

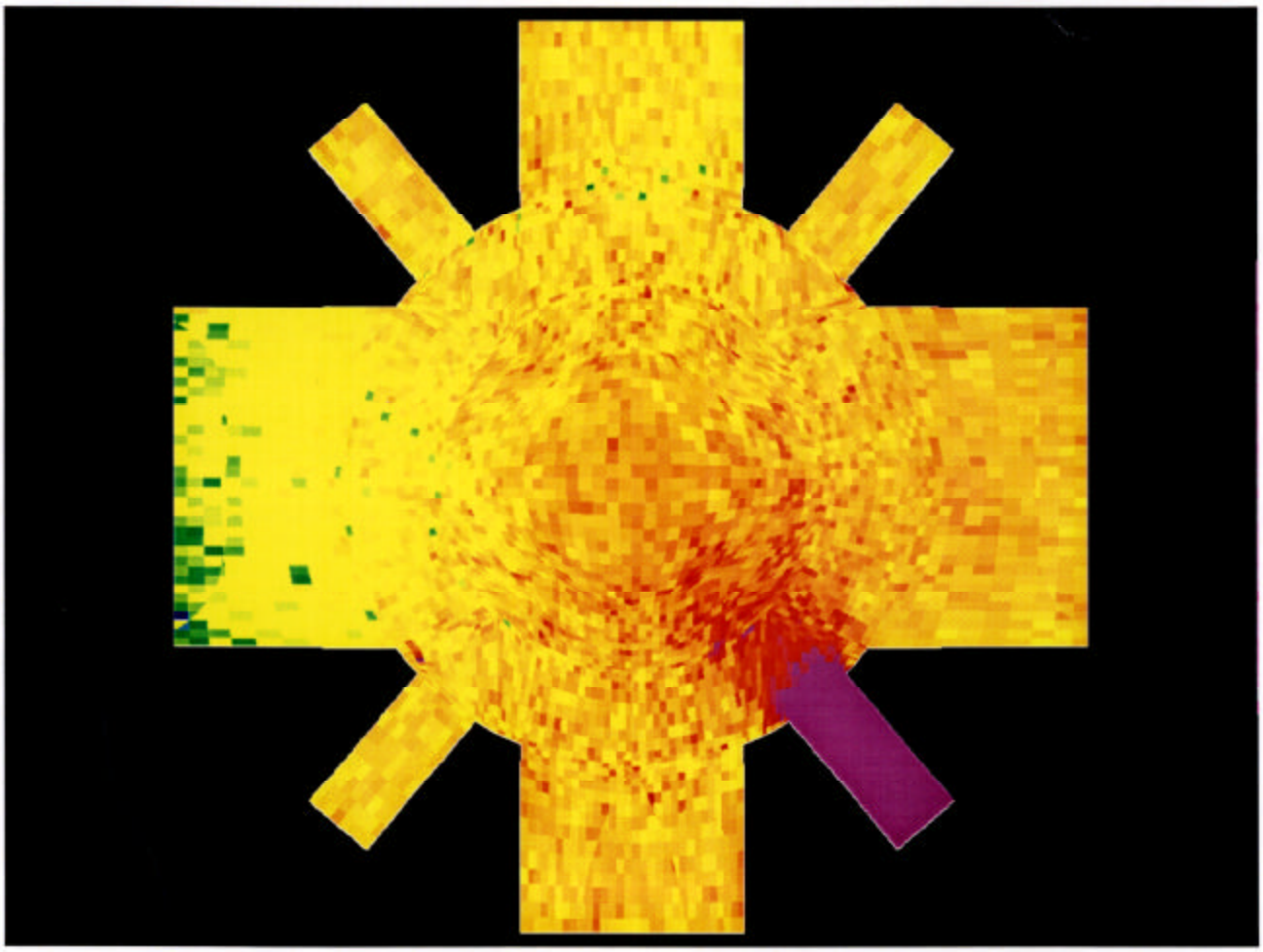

Figure 6.9: Density above the Wafer

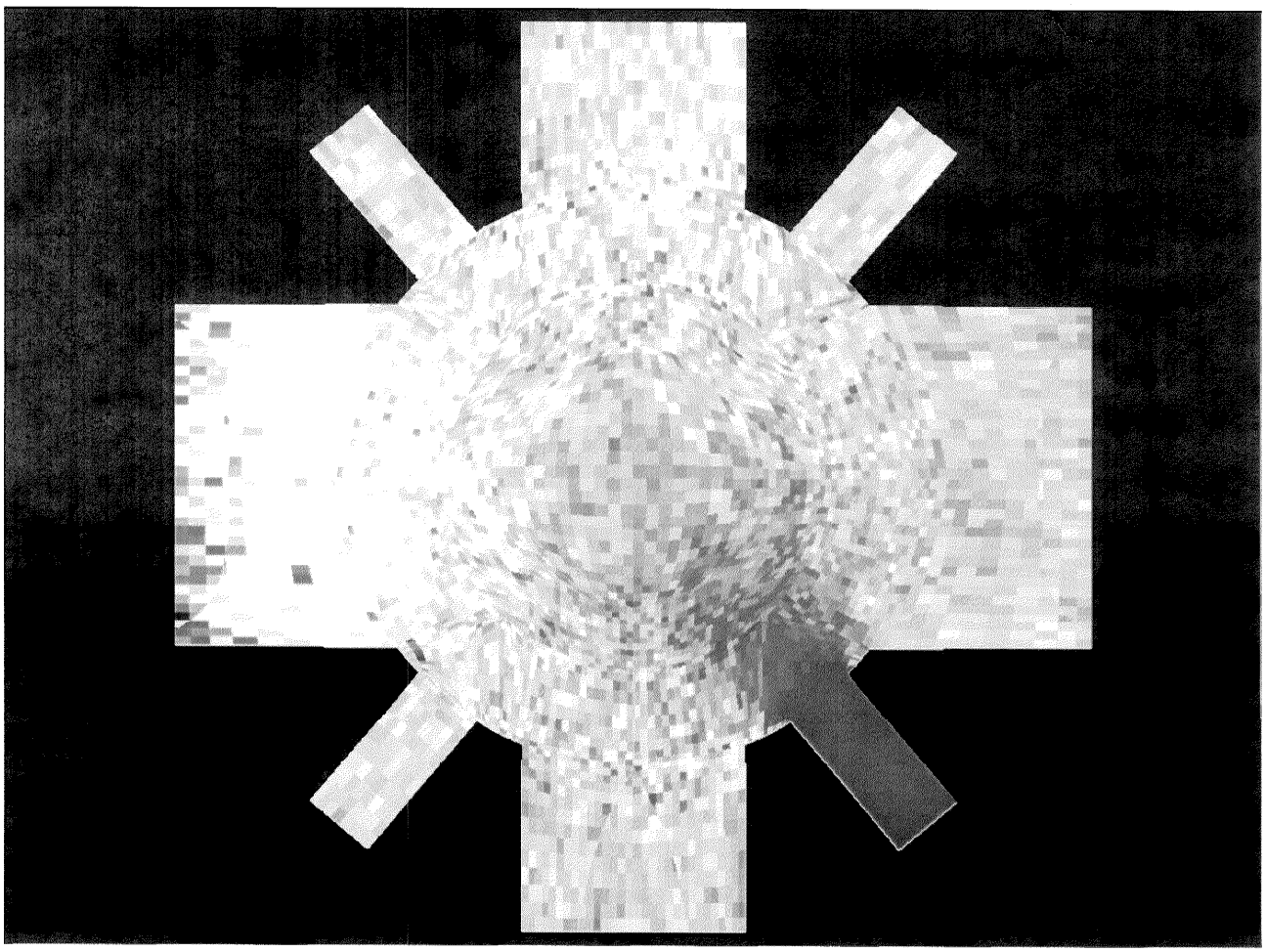

Figure 6.9: Density above the Wafer

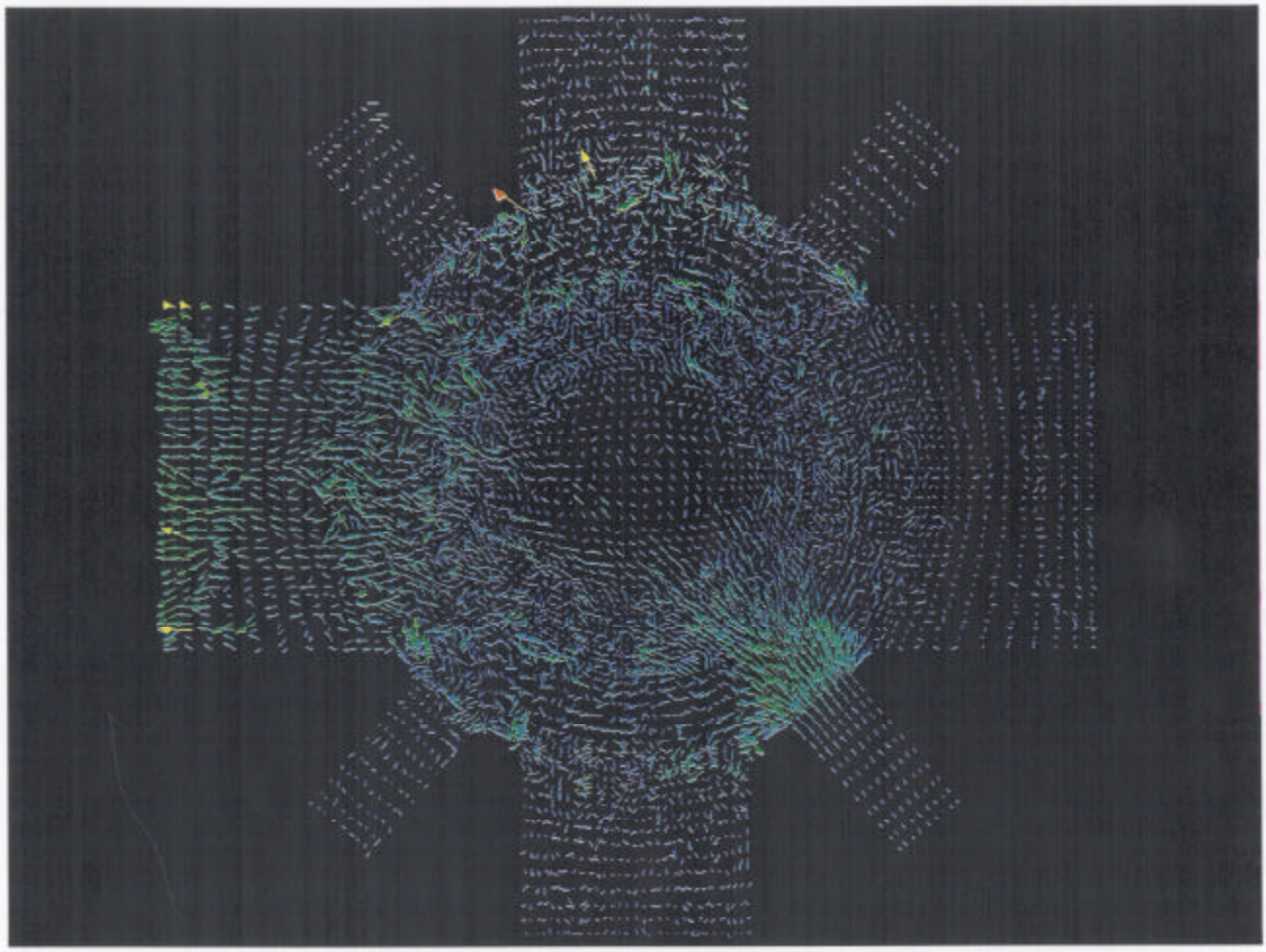

Figure 6.10: Velocity above the Wafer

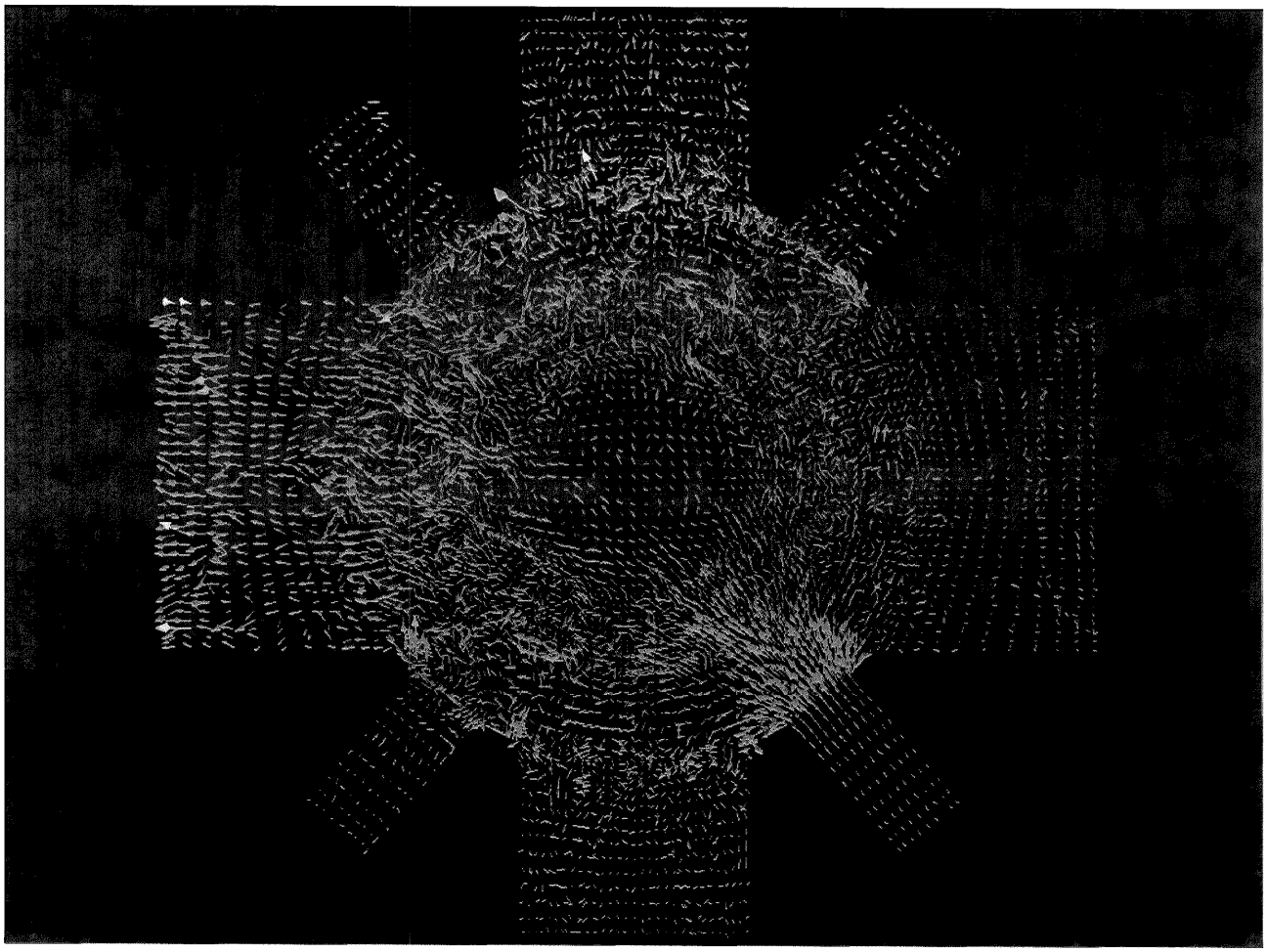

Figure 6.10: Velocity above the Wafer

6.3 Case II

In this case, each simulated particle represented 3×10^{11} real particles, yielding a total of about 500,000 simulated particles. The initial global timestep was 1×10^{-5} seconds, a value selected in order to minimize the number of steps required to obtain convergence.

Convergence History

Figure 6.11 shows the number of collisions between particles and walls for the entire domain. Significant fluctuation between timesteps is due to the statistical nature of the code, though a general exponential trend can be observed. Wall collisions appear to have reached a steady-state value by step 3,000. While it is clear that the simulation has not converged *before* step 3,000, this evidence is insufficient to conclude that the simulation is fully converged by step 3,000.

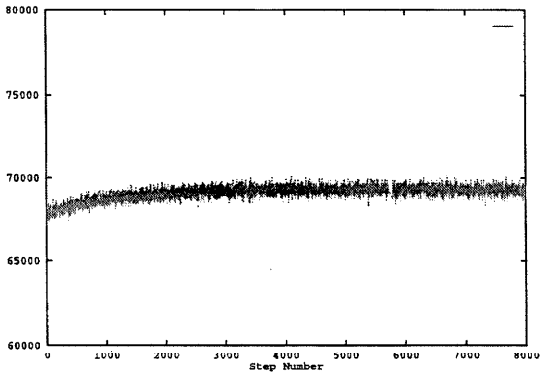

Figure 6.11: Wall Collisions

Figure 6.12 shows the number of particles in the domain as a function of timestep number. Since only a small fraction of the particles enter or leave the domain in a timestep, statistical fluctuations are small. Based on this graph, however, the simulation has clearly not converged before step 6,000. Similarly, Figure 6.13 shows the convergence history of the system's total kinetic energy. This result implies convergence around step 5,000.

It is clear that a number of macroscopic parameters must be studied in order to determine convergence. While another parameter might show a slightly longer convergence time, it is reasonable to conclude that the system has converged by step 8,000. The simulation was restarted at step 10,000, with a smaller timestep of 10^-6. The smaller timestep was selected so that particles would not travel too far at each step. If this timestep had been selected for the start of the simulation, convergence would have required approximately ten times as many steps. The simulation was run from step 10,000 to step 18,000 to allow any transients introduced by the change in timestep to decay. No change in global properties was observed during this period. At step 18,000, the statistics were reset. This ensured that sampling was only conducted over steady-state values. Sampling was conducted

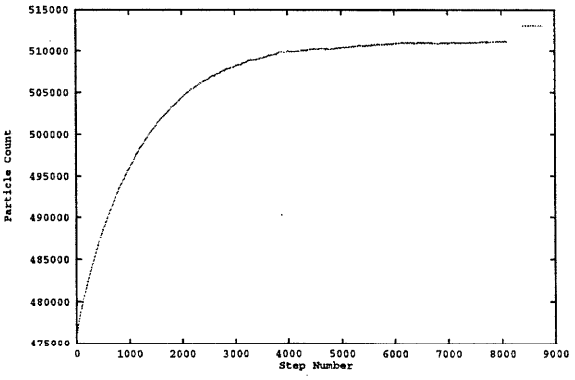

Figure 6.12: Particle Count

from step 18,000 until step 24,000 to obtain the smooth results presented in the next section.

Macroscopic Parameters

Figure 6.14 shows the density in a horizontal plane just above the wafer. The high inflow and low exhaust densities can be observed in this plot, just as in Figure 6.9. The most important difference between the two results, however, is in the differences in density between opposite ports. In Figure 6.9, there is only a slight density difference between the large port on the right and that on the left, and there is no apparent difference between top and bottom ports. In Figure 6.14, however, there is a distinct increase in density from left to right, and the density in the top port is noticeably lower than that in the bottom port. This is because the simulation in Figure6.14 has completely converged; it has had time to establish gentle gradients throughout the reactor.

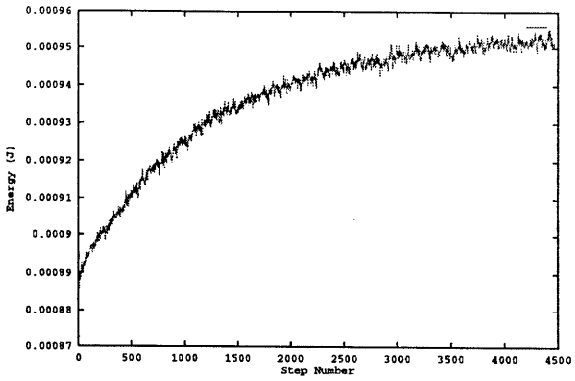

Figure 6.13: System Kinetic Energy

Figure 6.15 shows the average velocity of gas particles within each cell. As in Figure 6.10, the inflow and exhaust ports are apparent, and the flow over the wafer is slow. The most obvious difference between the two is in the amount of statistical scatter. This is most notable in the exhaust port, where Figure 6.15 shows uniform parallel flow, while Figure 6.10 shows some non-uniformity and flow that is not always parallel. Perhaps more importantly, though, the flow around the wafer is distinctly different between the two figures. Figure 6.10 shows disorganized flow in all directions, while Figure 6.15 shows very little net flow. The reason for the discrepancy is that results in Figure 6.10 have not been averaged for as many steps as those in Figure 6.15.

The comparison between these two cases emphasizes the importance of a detailed sensitivity analysis. In order to safely conclude the validity of these simulation results, more detailed parametric studies will be performed in the near future. Simulations must be conducted with different numbers of simulated particles to verify that sufficient number has been used. Different timesteps should be selected to isolate timestep dependences. Different initial conditions should be examined to ensure that they do not affect the solution. Finally, simulations should be performed with varying grid resolutions. Once these tests have been concluded, simulation results should be compared with experimental data.

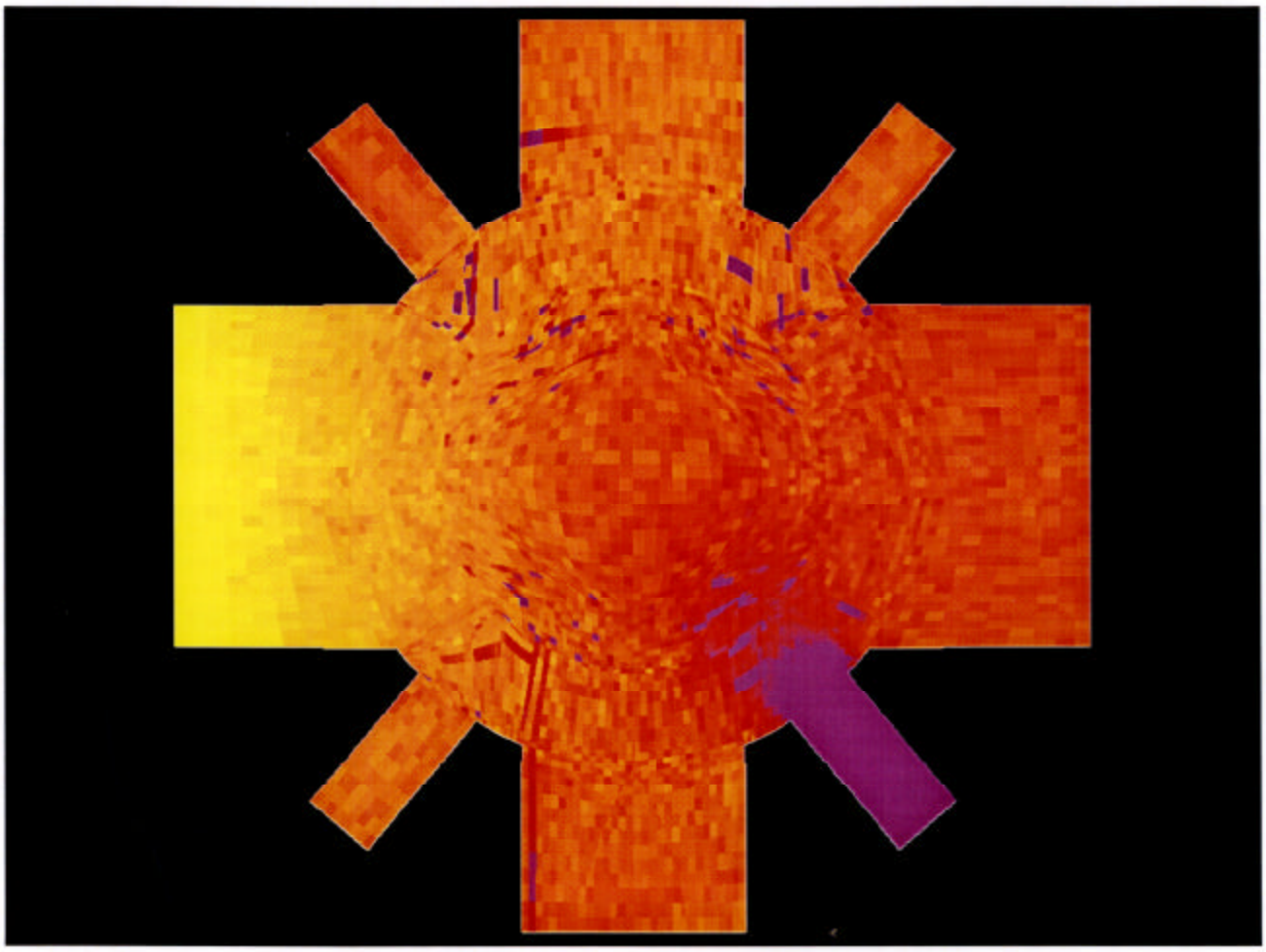

Figure 6.14: Density above the Wafer

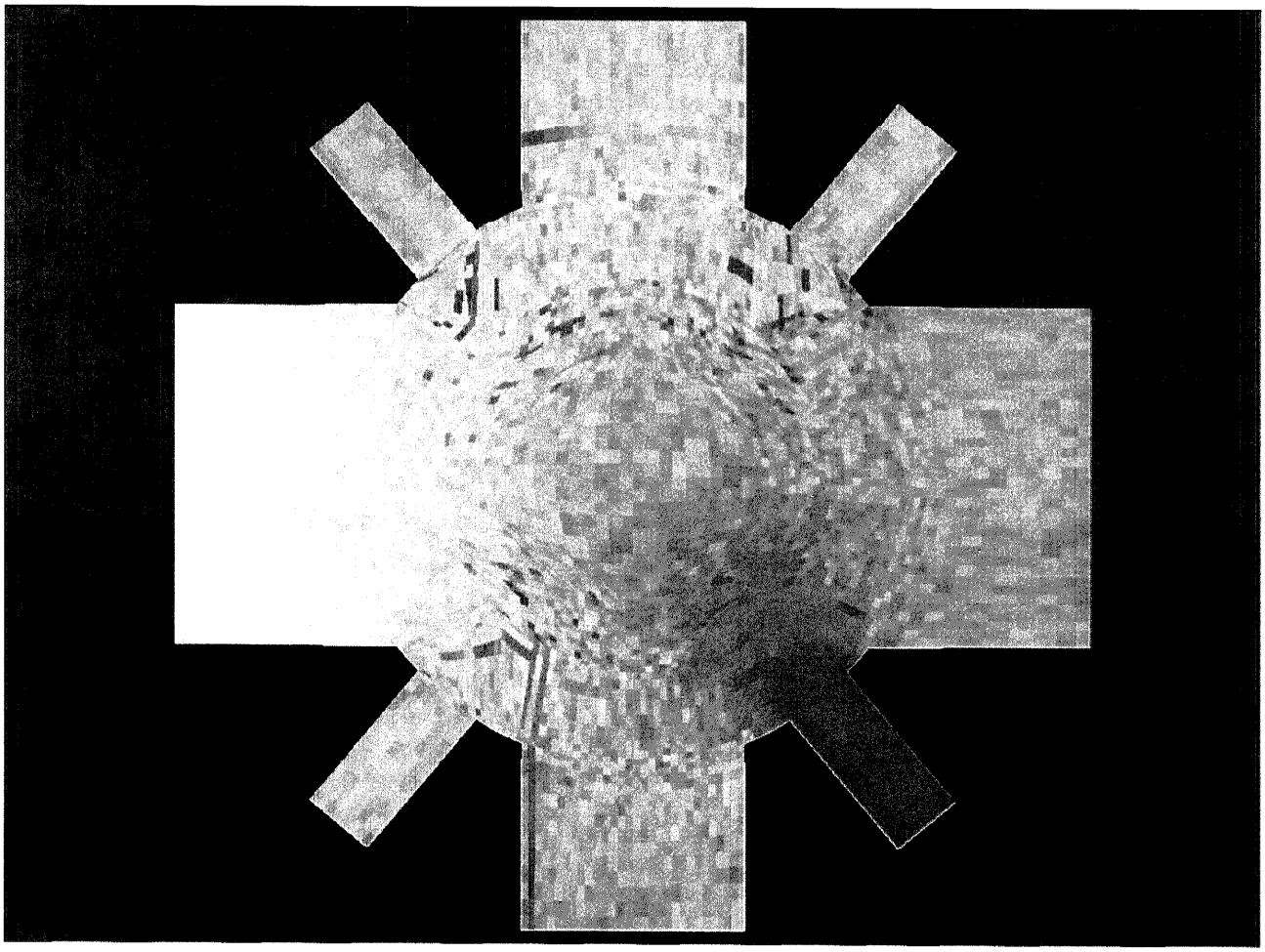

Figure 6.14: Density above the Wafer

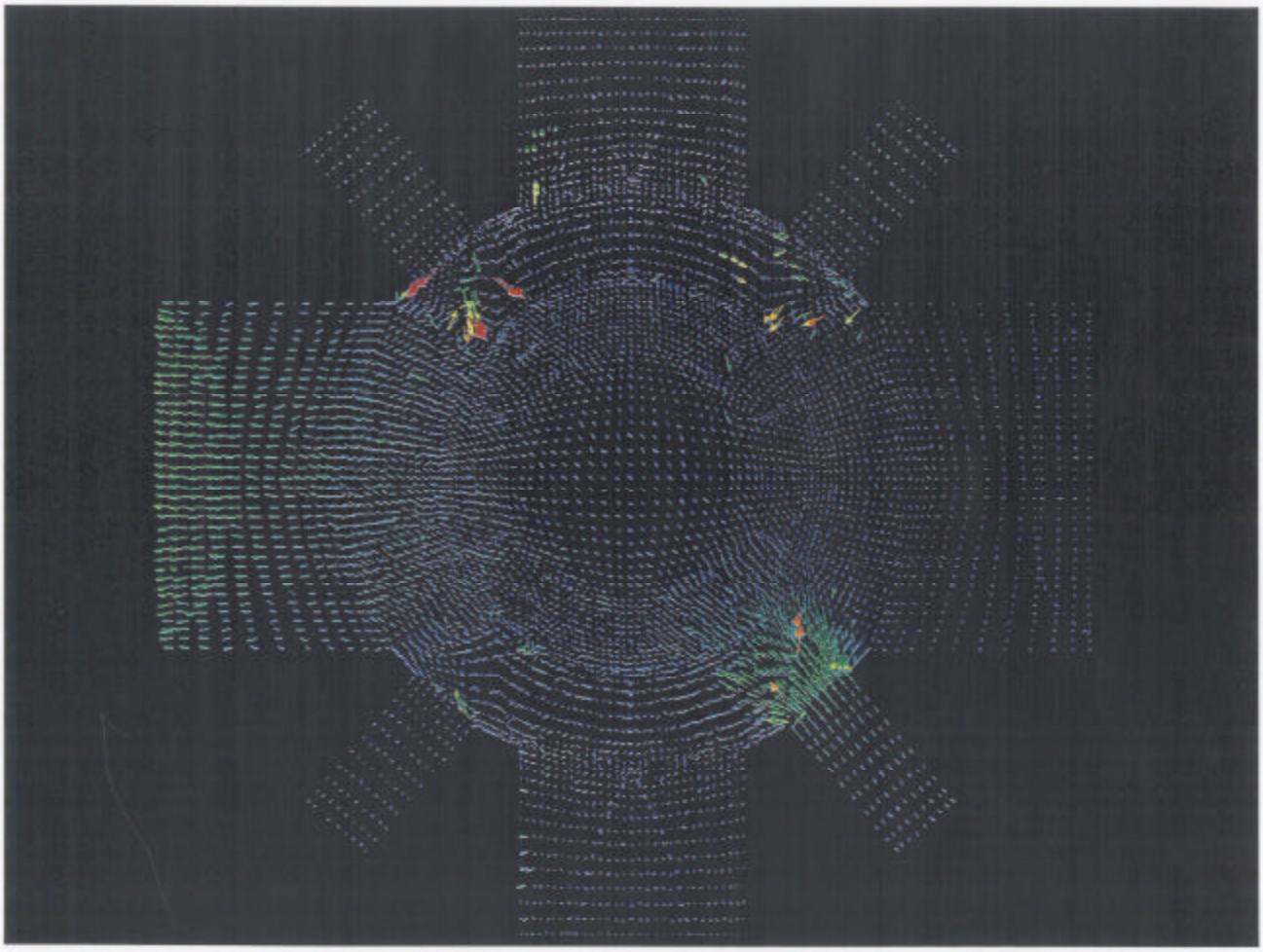

Figure 6.15: Velocity above the Wafer

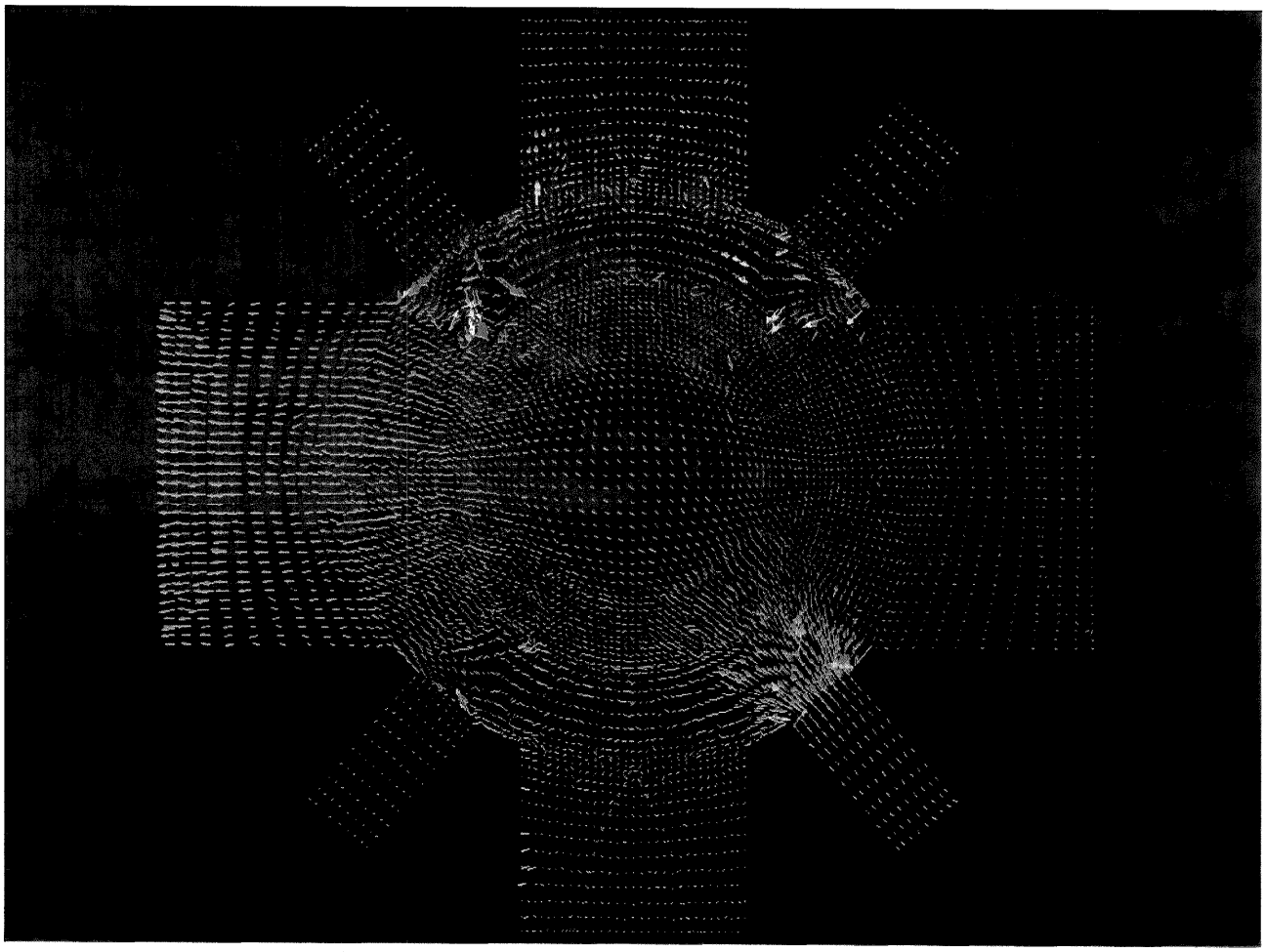

Figure 6.15: Velocity above the Wafer

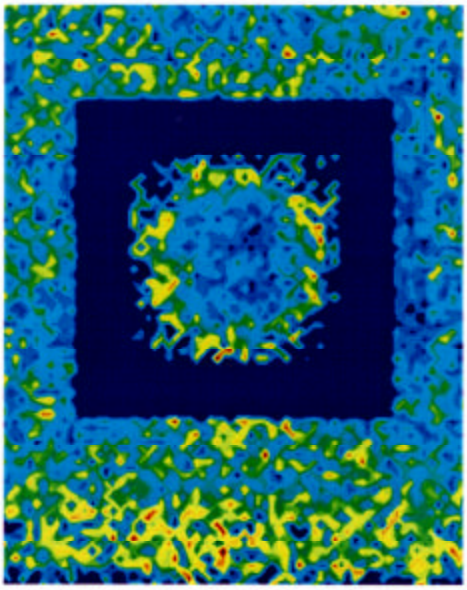

Figure 6.16: Intel Reactor Results

6.4 Intel-Proprietary Simulations

Hawk has also been used by Intel to simulate to simulate neutral flow in a proprietary reactor on 64 nodes of an Intel Paragon. Figure 6.16 shows particle speed above the wafer for this simulation. The wafer is the round region in the center of the plot. These results agree qualitatively with experimental observations. Because of the proprietary nature of these simulations, no further details can be presented.

6.5 Load Balancing

One would expect that a standard spatial decomposition and mapping of the grid would result in a very inefficient computation. In order to verify this, the GEC grid was divided into 2,560 partitions and mapped onto a 256-processor Intel Paragon. Because of the wide variance in particle density for each partition, the overall efficiency of the computation was quite low, at approximately 11 percent. As shown in Figure 6.17, this efficiency was improved to 86 percent by load balancing. This resulted in an 88 percent reduction in the run time. Figure 6.18 shows the corresponding improvement in workload distribution. The benefits of load balancing are expected to be much more dramatic for larger numbers of processors, and for networks of workstations that are in use.

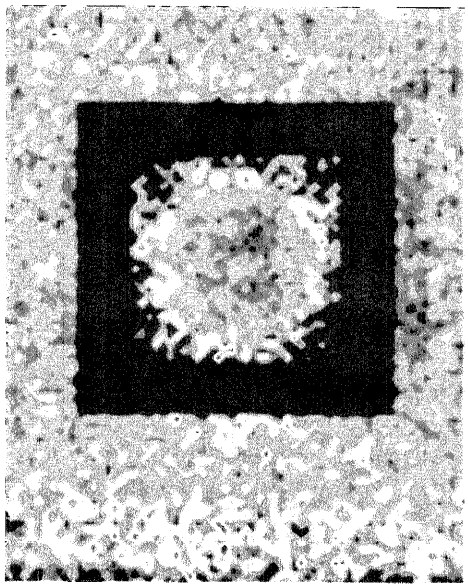

Figure 6.16: Intel Reactor Results

6.4 Intel-Proprietary Simulations

Hawk has also been used by Intel to simulate to simulate neutral flow in a proprietary reactor on 64 nodes of an Intel Paragon. Figure 6.16 shows particle speed above the wafer for this simulation. The wafer is the round region in the center of the plot. These results agree qualitatively with experimental observations. Because of the proprietary nature of these simulations, no further details can be presented.

6.5 Load Balancing

One would expect that a standard spatial decomposition and mapping of the grid would result in a very inefficient computation. In order to verify this, the GEC grid was divided into 2,560 partitions and mapped onto a 256-processor Intel Paragon. Because of the wide variance in particle density for each partition, the overall efficiency of the computation was quite low, at approximately 11 percent. As shown in Figure 6.17, this efficiency was improved to 86 percent by load balancing. This resulted in an 88 percent reduction in the run time. Figure 6.18 shows the corresponding improvement in workload distribution. The benefits of load balancing are expected to be much more dramatic for larger numbers of processors, and for networks of workstations that are in use.

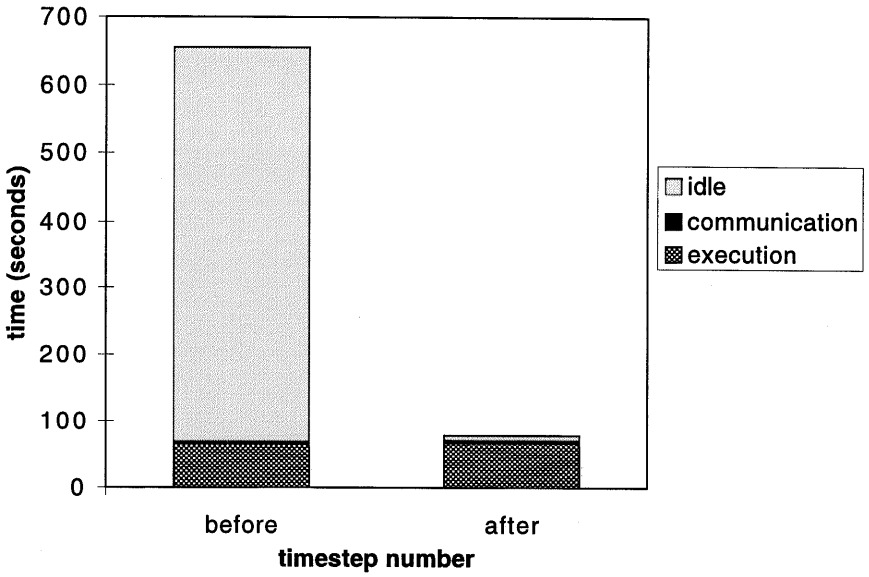

Figure 6.17: Run Time Distribution

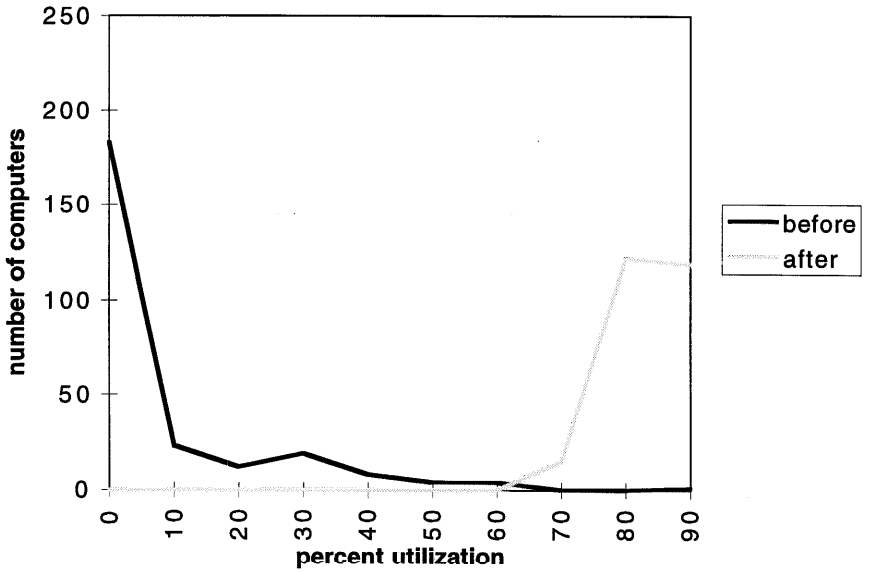

Figure 6.18: Load Distribution

Chapter 7

Conclusion

The preceding chapters have presented the basic concepts of the DSMC method, the details of *Hawk*'s implementation, experiments conducted to validate *Hawk* 's physics, and the results of large-scale simulations. While much progress has been made, this work represents only the beginning of a comprehensive plasma-simulation project. Many more features must be added, new models developed, and optimizations implemented, before *Hawk* will be capable of complete simulations of plasma processes.

Planned improvements to *Hawk* fall into two main categories: improved chemical and physical models, and new computational techniques to improve the speed and accuracy of existing models. Decreasing the time for complete simulations can have tremendous benefit, greatly enhancing the applicability of simulation tools.

With the current system, a complete simulation requires about a week of effort to generate a grid, and a week to run the simulation. The time required to generate a computational grid can be drastically reduced by switching to automatically-generated unstructured grids. For example, it took more than a week to generate a structured hexahedral grid for the GEC reference cell. By contrast, it was possible to produce an unstructured hexahedral grid in less than two days, and an unstructured tetrahedral grid in an afternoon. While most of the underlying algorithms in *Hawk*'s implementation are unstructured in nature, *Hawk* does not yet support simulations on unstructured grids. Switching *Hawk* to be fully unstructured, and to support unstructured grids of arbitrary polyhedra, is an obvious opportunity for reducing the total time necessary to obtain a simulation result.

The inclusion of ions and electrons in simulations will significantly enhance *Hawk*'s ability to model reactor chemistry. Because electrons move much faster than other species, they must be simulated at a much smaller timestep. Several methods are under consideration that will permit each species to be moved at on a different timescale. Inclusion of electrons, however, is sure to substantially increase the complexity of a simulation.

The self-consistent solution of electromagnetic fields will be necessary for modeling electron and ion flow in reactors. Several techniques, including the boundary-element method (BEM) and particle-in-cell (PIC) method, are being evaluated for this purpose. It is estimated that incorporation of electromagnetic fields could as much as double the computational requirements of a simulation.

In order to obtain good DSMC simulation results, as a rule of thumb, the Knudsen number in each cell must be at least 3. Currently, the only way to ensure this is trial and error. An initial grid can be generated based on expectations about the flow patterns. If certain areas of the grid are too coarse, a new grid must be generated, and a new simulation performed. With each simulation requiring days of computation, this is a very expensive cycle. The only way that one could generate an acceptable grid the first time would be if the flow patterns were already known, in which case no simulation would be necessary. An adaptive solver, on the other hand, could split and combine grid cells based on cell Knudsen numbers or other flow features, ensuring an efficient and correct grid on every iteration. Since only a rough initial grid would be necessary, this would also help to decrease grid generation time.

Another opportunity for grid adaption is in the calculation of macroscopic parameters. The current implementation computes results in each cell. In some regions of the flow, collision constraints may require many small cells, each of which is too small to be observed in a graph. An adaptive, hierarchical grid system would compute results for aggregates of such cells if they had similar properties. Similarly, large cells containing significant gradients could be split to improve the resolution of the results. These improvements could significantly improve the simulation quality and reduce simulation time.

While the main goal of this work is to support the microelectronics industry, the generality of the tool allows it to be applied to other areas as well. The DSMC method is particularly applicable to satellite problems; *Hawk* has already been used to simulate the reentry of the Skipper satellite. Once self-consistent electromagnetic fields have been incorporated, it will also be useful for backflow contamination and other free-molecular problems such as have been traditionally simulated using the Particle-In-Cell technique [Roy95].

The results of this work show that large-scale simulations of realistic plasma reactors are possible. For realistic simulations, two-dimensional axisymmetric calculations are insufficient. In order to support complex geometries and adaptation, dynamic data structures are necessary. Simulation of large systems requires extensive computational resources, necessitating the use of modern concurrent architectures and networks of workstations. Efficient parallel implementations require the use of modern computer science and software engineering techniques.

While only single-species neutral flow has been presented and validated here, significant progress has been made in support of multiple species, complex inelastic collisions and reactions, self-consistent electromagnetic fields, and adaptive grids. A result of this work has been to establish the *software infrastructure* that is necessary to incorporate these features. After a methodical series of validation experiments, *Hawk* will be capable of accurate simulation of all relevant processes inside a plasma reactor. This will allow industrial designers and process engineers to evaluate new reactor designs and configurations quickly, accurately, and cost-effectively.

Appendix A

Using *Hawk*

After a grid has been generated, several simulation configuration steps are necessary. Two graphical tools, *XHawk* and *XFalcon*, have been developed to simplify this process. *XHawk* is used to configure simulations, and *XFalcon* is used to configure chemistry databases. Chemistry information, such as species mass and reaction cross sections, is stored in a database and manipulated with *XFalcon*. Simulation parameters, such as the species used in the simulation, the global timestep, and F_N are specified in the *XHawk* configuration program. Once a set of species has been selected for a simulation, *XHawk* extracts the appropriate chemistry information from the *XFalcon* database, and stores all of *Hawk*'s input parameters in a settings file, as shown in Figure A.1. The following sections describe the *XHawk* and *XFalcon* utility programs.

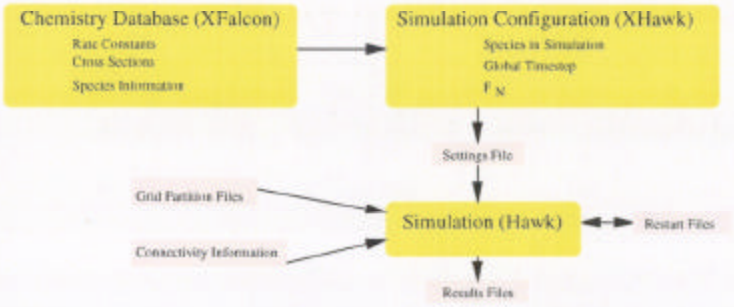

Figure A.1: Simulation Configuration

A.1 Simulation Configuration with *XHawk*

Hawk obtains configuration information from a *settings file*. This settings file is maintained in a human readable form, but is typically edited using *XHawk* . By hierarchically organizing configuration parameters, *XHawk* greatly simplifies simulation preparation.

Some of the information in the settings file includes, the location of the grid that defines the simulation, the species used and their reactivity properties, initial conditions, surface properties, the number of computers to be used, and the number of steps during which samples should be accumulated.

XHawk is organized in such a way to allow for settings files to be collaboratively prepared by different users. For example, chemistry information might be specified by a process engineer,

Appendix A

Using *Hawk*

After a grid has been generated, several simulation configuration steps are necessary. Two graphical tools, *XHawk* and *XFalcon*, have been developed to simplify this process. *XHawk* is used to configure simulations, and *XFalcon* is used to configure chemistry databases. Chemistry information, such as species mass and reaction cross sections, is stored in a database and manipulated with *XFalcon*. Simulation parameters, such as the species used in the simulation, the global timestep, and F_N are specified in the *XHawk* configuration program. Once a set of species has been selected for a simulation, *XHawk* extracts the appropriate chemistry information from the *XFalcon* database, and stores all of *Hawk*'s input parameters in a settings file, as shown in Figure A.1. The following sections describe the *XHawk* and *XFalcon* utility programs.

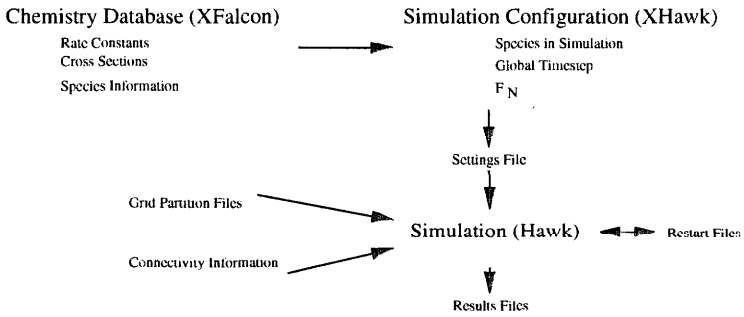

Figure A.1: Simulation Configuration

A.1 Simulation Configuration with *XHawk*

Hawk obtains configuration information from a *settings file*. This settings file is maintained in a human readable form, but is typically edited using *XHawk* . By hierarchically organizing configuration parameters, *XHawk* greatly simplifies simulation preparation.

Some of the information in the settings file includes, the location of the grid that defines the simulation, the species used and their reactivity properties, initial conditions, surface properties, the number of computers to be used, and the number of steps during which samples should be accumulated.

XHawk is organized in such a way to allow for settings files to be collaboratively prepared by different users. For example, chemistry information might be specified by a process engineer,

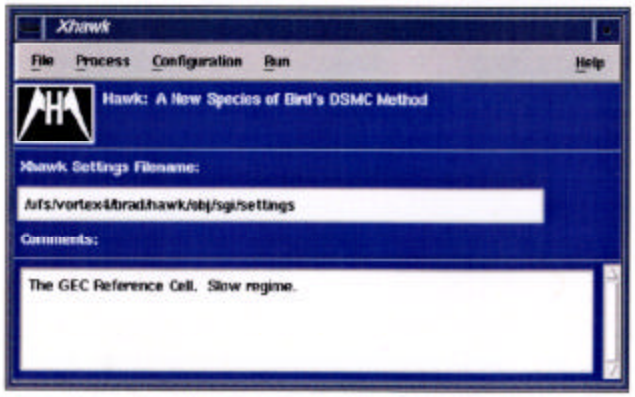

Figure A.2: Main *XHawk* window

while the particular properties of how the simulation is to be run on its target system, may need to be adjusted by the system administrator.

XHawk is an X windows application that uses the Motif widget set. It is designed to be portable and has been tested on a variety of X- Windows based systems including Suns, IBMs, and SGIs. *XHawk* has been created primarily using C++. A C++ wrapper has been designed, encapsulating all X windows code, and is maintained separately from the rest of the application. This same C++ interface to X windows is also used by *XFalcon* .

Simulation Configuration

Figure A.2 shows *XHawk* 's main window. Menu options are used to create new settings files and to edit old ones. The text field at the center is used to textually describe the simulation being configured.

XHawk provides options allowing the selection of the chemical model that will be used in the current simulation. Selecting the "Chemistry" option from the "Process" menu activates the dialog box shown in Figure A.3. Selecting a species in this window activates the species definition box, shown in Figure A.4, where the user can specify the name, mass, and charge of a species. Selecting a reaction from the list allows the user to specify the details of a reaction, as shown in Figure A.5.

By selecting the "execution" option in the main window, the dialog box shown in Figure A.6 is activated. Here, users select execution parameters such as the number of processors to use and how the grid will be partitioned for distribution between the processors.

Statistical information, such as the number of steps over which to accumulate statistics, how often to write or reset statistics, and the global timestep, can be specified as shown in Figure A.7.

XHawk allows the user to select surface and volume properties, as shown in Figure A.8. Surface types, such as inflow, outflow, and solid, are configured in the dialog box shown in Figure A.9. If appropriate, surface properties, such as inflow density and temperature, are also specified there.

Initial conditions, such as density and temperature, are configured as shown in Figure A.10.

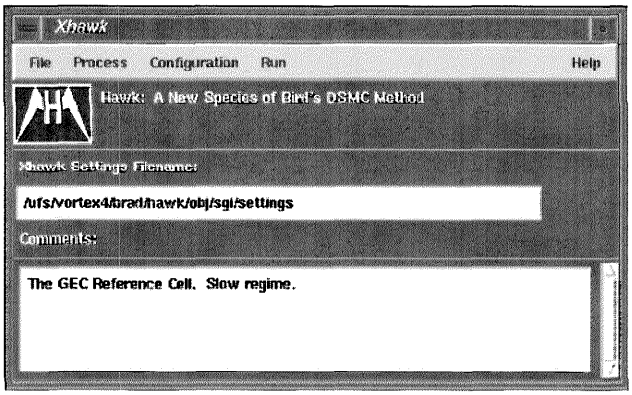

Figure A.2: Main *XHawk* window

while the particular properties of how the simulation is to be run on its target system, may need to be adjusted by the system administrator.

XHawk is an X windows application that uses the Motif widget set. It is designed to be portable and has been tested on a variety of X- Windows based systems including Suns, IBMs, and SGIs. *XHawk* has been created primarily using C++. A C++ wrapper has been designed, encapsulating all X windows code, and is maintained separately from the rest of the application. This same C++ interface to X windows is also used by *XFalcon* .

Simulation Configuration

Figure A.2 shows *XHawk* 's main window. Menu options are used to create new settings files and to edit old ones. The text field at the center is used to textually describe the simulation being configured.

XHawk provides options allowing the selection of the chemical model that will be used in the current simulation. Selecting the "Chemistry" option from the "Process" menu activates the dialog box shown in Figure A.3. Selecting a species in this window activates the species definition box, shown in Figure A.4, where the user can specify the name, mass, and charge of a species. Selecting a reaction from the list allows the user to specify the details of a reaction, as shown in Figure A.5.

By selecting the "execution" option in the main window, the dialog box shown in Figure A.6 is activated. Here, users select execution parameters such as the number of processors to use and how the grid will be partitioned for distribution between the processors.

Statistical information, such as the number of steps over which to accumulate statistics, how often to write or reset statistics, and the global timestep, can be specified as shown in Figure A.7.

XHawk allows the user to select surface and volume properties, as shown in Figure A.8. Surface types, such as inflow, outflow, and solid, are configured in the dialog box shown in Figure A.9. If appropriate, surface properties, such as inflow density and temperature, are also specified there.

Initial conditions, such as density and temperature, are configured as shown in Figure A.10.

Figure A.3: Chemistry Configuration

Figure A.4: Species Definition

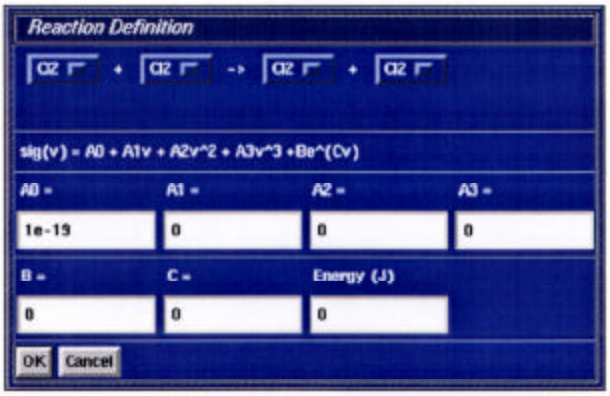

Figure A.5: Reaction Definition

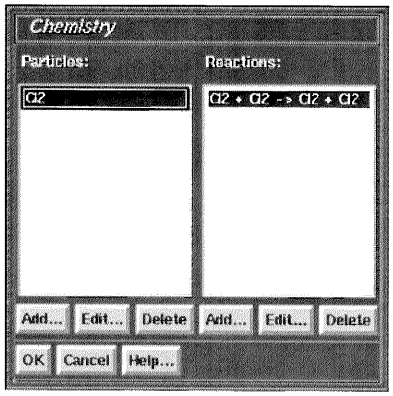

Figure A.3: Chemistry Configuration

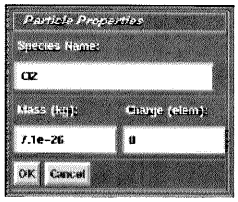

Figure A.4: Species Definition

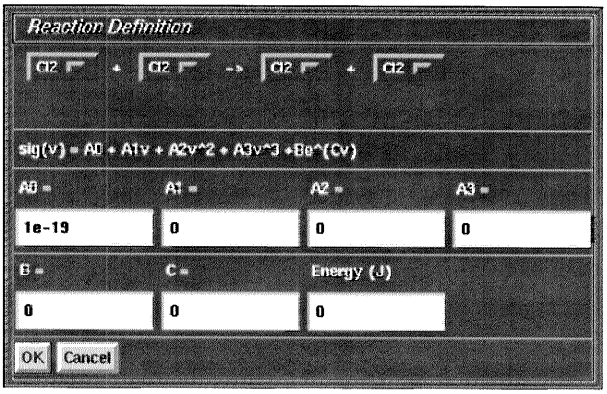

Figure A.5: Reaction Definition

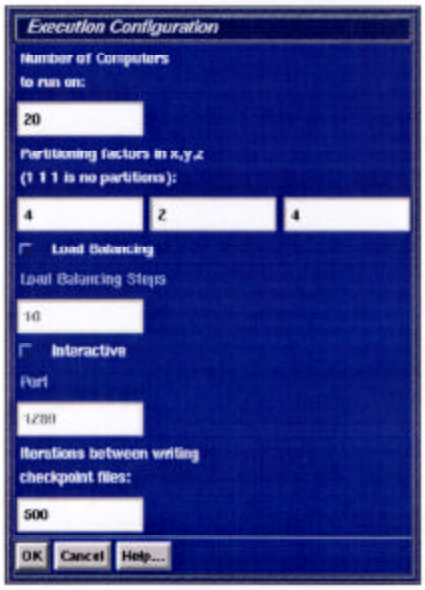

Figure A.6: Execution Configuration

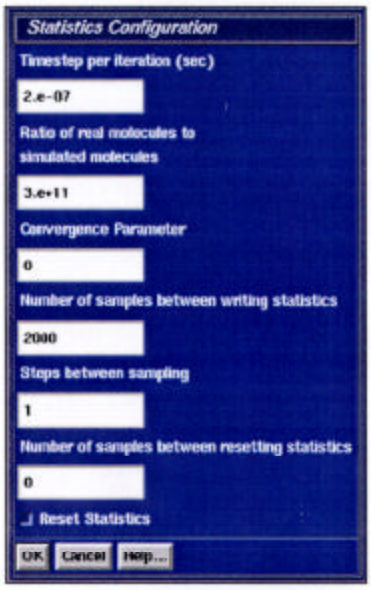

Figure A.7: Statistics Configuration

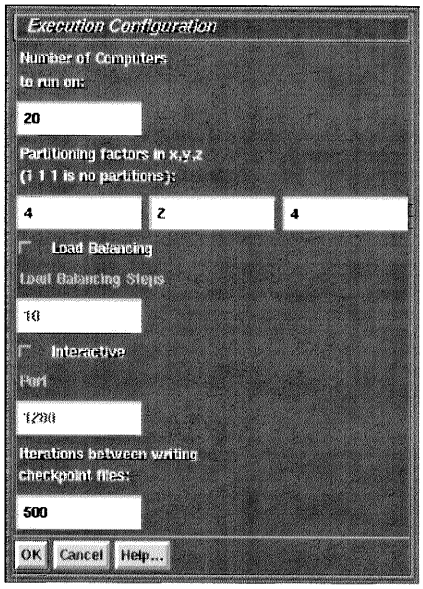

Figure A.6: Execution Configuration

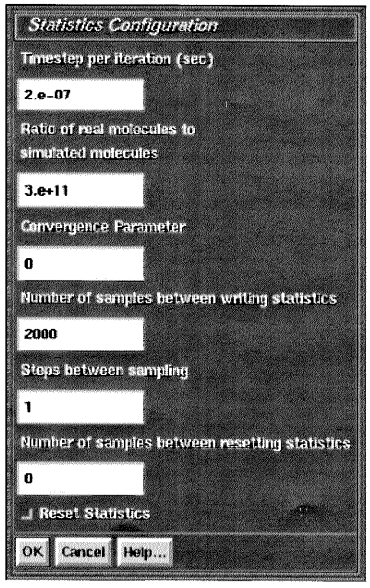

Figure A.7: Statistics Configuration

Figure A.8: Surface and Volume Specification

Figure A.9: Surface Type Configuration

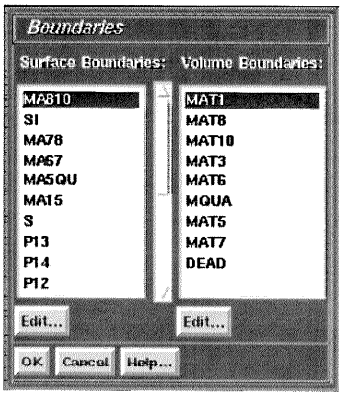

Figure A.8: Surface and Volume Specification

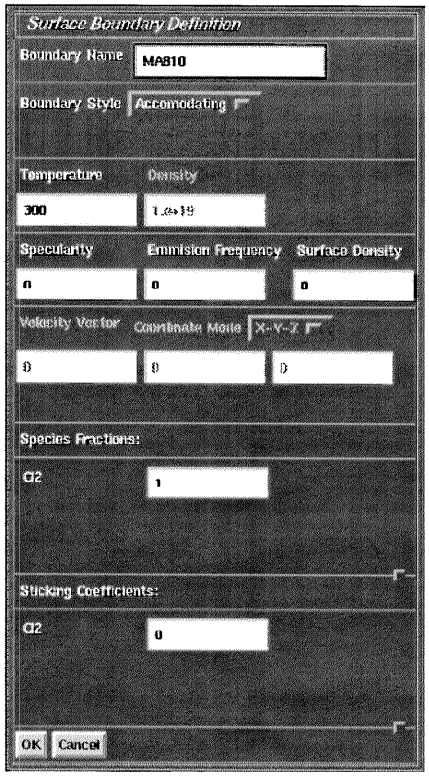

Figure A.9: Surface Type Configuration

Figure A.10: Initial Condition Specification

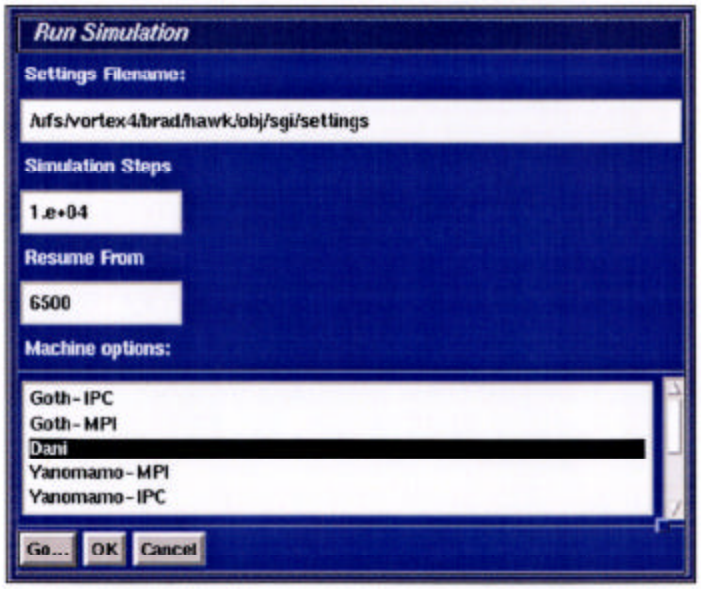

Figure A.11: Simulation Execution

Interactive Simulation Control

XHawk provides various options to facilitate interactive simulation execution. In other words, a user can configure a simulation using *XHawk* on a workstation, and instruct *XHawk* to initiate a simulation on a remote parallel machine. During the execution of the simulation, graphical feedback of the simulation's progress is displayed on the user's workstation. This set of features reduces the complexities associated with starting and stopping jobs on parallel computers, and does not require the user to have any knowledge about parallel computers.

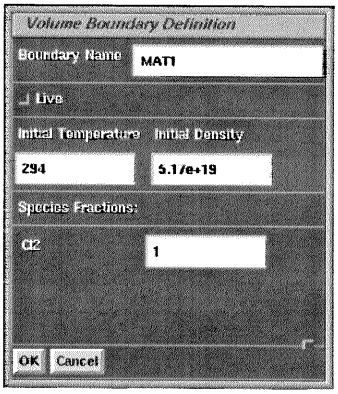

Figure A.10: Initial Condition Specification

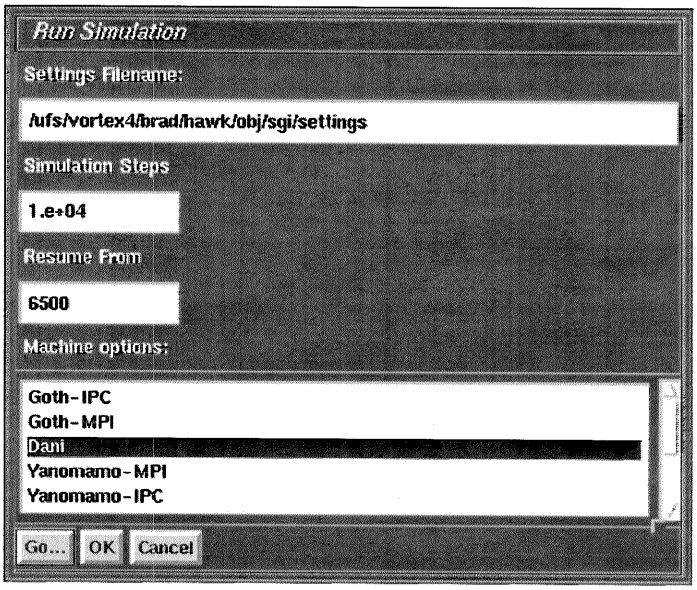

Figure A.11: Simulation Execution

Interactive Simulation Control

XHawk provides various options to facilitate interactive simulation execution. In other words, a user can configure a simulation using *XHawk* on a workstation, and instruct *XHawk* to initiate a simulation on a remote parallel machine. During the execution of the simulation, graphical feedback of the simulation's progress is displayed on the user's workstation. This set of features reduces the complexities associated with starting and stopping jobs on parallel computers, and does not require the user to have any knowledge about parallel computers.

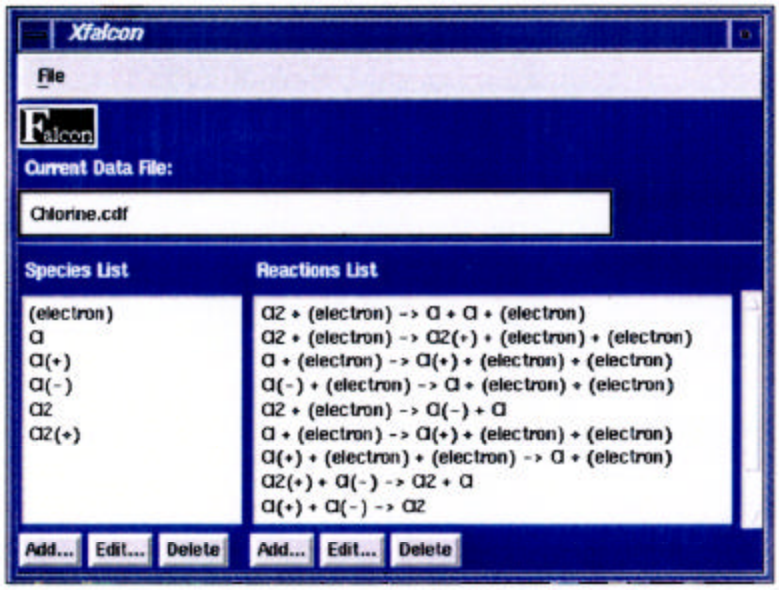

Figure A.12: Database Configuration

A.2 Chemistry Database Manipulation with *XFalcon*

A database system, *Falcon* has been created to allow users to maintain descriptions of chemical reactions that might be useful in applying *Hawk* to real world simulations. The data it will contain consists of certain fundamental physical properties of known chemical species types as well as information on the various reactions that can occure involving these species. *XFalcon* is used to create and maintain *Falcon* databases.

Falcon databases are designed on top of GNU's GDBM system in order to to provide the equivalent of a disk based memory allocation system. GDBM allows allocation of disk blocks and maintains a key to each allocation that can be used as a pointer, facilitation the implementation of disk based trees. Such trees are then used to maintain the data associatcd with each species and its reactions. The database format is designed in such a way that new data elements can be added without introducing incompatibilities with existing application. This is accomplished by various markers indicating version number as well as elements that contain counts of non-descipt sets of data. SCP's extio library [Watts95a] is used in conjunction with the GDBM system to allow storage to disk that is independent of machine word size.

XFalcon presently supports an atomic content description of the species it contains. Reactions are defined in terms of the species involved, and in terms of intervals of velocity dependant cross-section information. This information can be input in a variety of forms, but is internally maintained in a universal format.

The primary view of a *Falcon* database that is provided by *XFalcon* shows known species and reactions, as shown in Figure A.12. These lists can be updated to reflect new chemical information. Species are defined in terms of their atomic content, their charge, their mass, and are uniquely identified in terms of their excited state by their extended name. See Figure A.13.

Reactions are uniquely identified by the reactants and products involved in a reaction, as

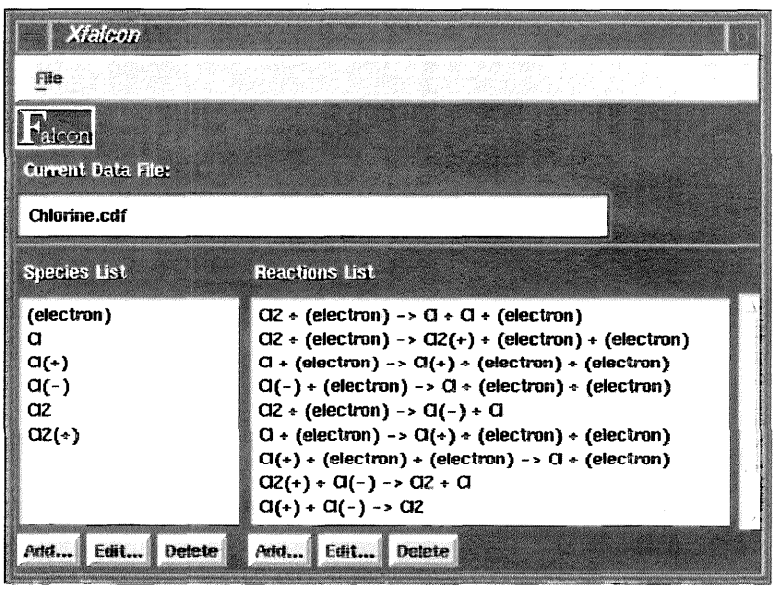

Figure A.12: Database Configuration

A.2 Chemistry Database Manipulation with *XFalcon*

A database system, *Falcon* has been created to allow users to maintain descriptions of chemical reactions that might be useful in applying *Hawk* to real world simulations. The data it will contain consists of certain fundamental physical properties of known chemical species types as well as information on the various reactions that can occure involving these species. *XFalcon* is used to create and maintain *Falcon* databases.

Falcon databases are designed on top of GNU's GDBM system in order to to provide the equivalent of a disk based memory allocation system. GDBM allows allocation of disk blocks and maintains a key to each allocation that can be used as a pointer, facilitation the implementation of disk based trees. Such trees are then used to maintain the data associated with each species and its reactions. The database format is designed in such a way that new data elements can be added without introducing incompatibilities with existing application. This is accomplished by various markers indicating version number as well as elements that contain counts of non-descipt sets of data. SCP's extio library [Watts95a] is used in conjunction with the GDBM system to allow storage to disk that is independent of machine word size.

XFalcon presently supports an atomic content description of the species it contains. Reactions are defined in terms of the species involved, and in terms of intervals of velocity dependant cross-section information. This information can be input in a variety of forms, but is internally maintained in a universal format.

The primary view of a *Falcon* database that is provided by *XFalcon* shows known species and reactions, as shown in Figure A.12. These lists can be updated to reflect new chemical information. Species are defined in terms of their atomic content, their charge, their mass, and are uniquely identified in terms of their excited state by their extended name. See Figure A.13.

Reactions are uniquely identified by the reactants and products involved in a reaction, as

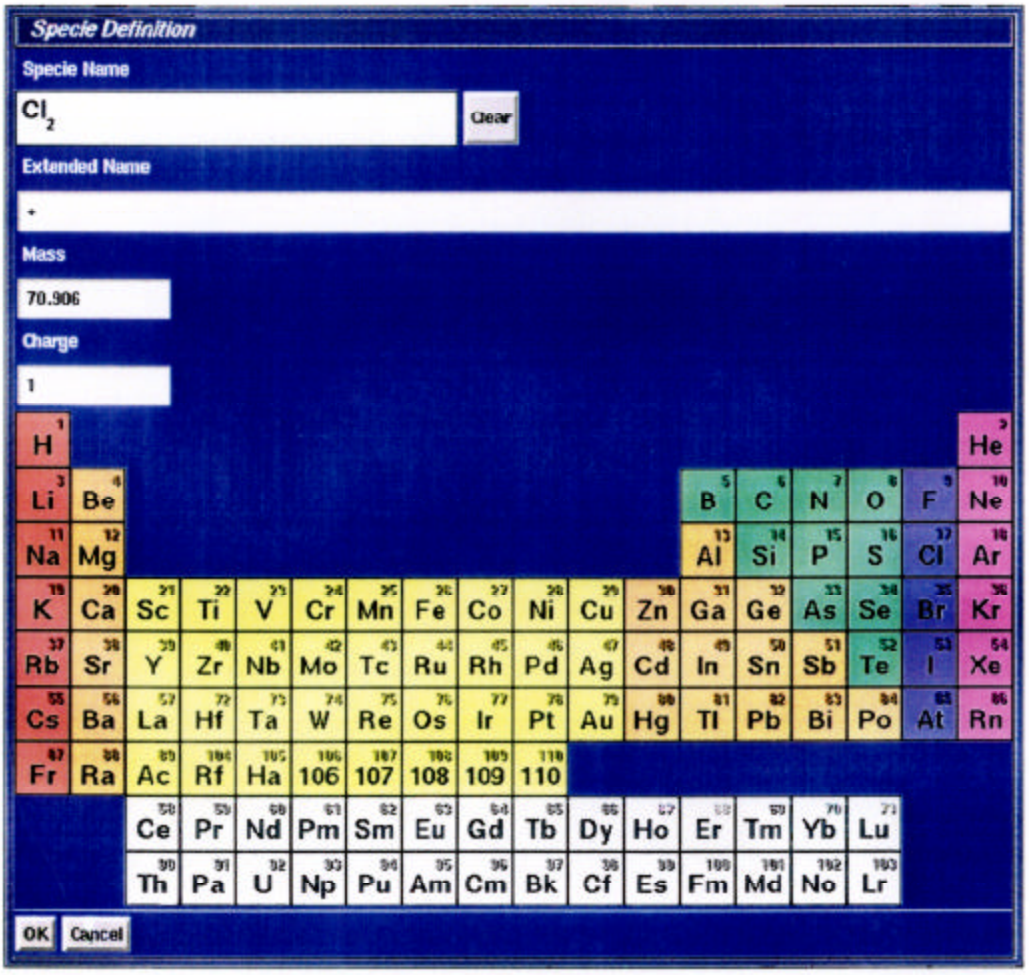

Figure A.13: Species Definition

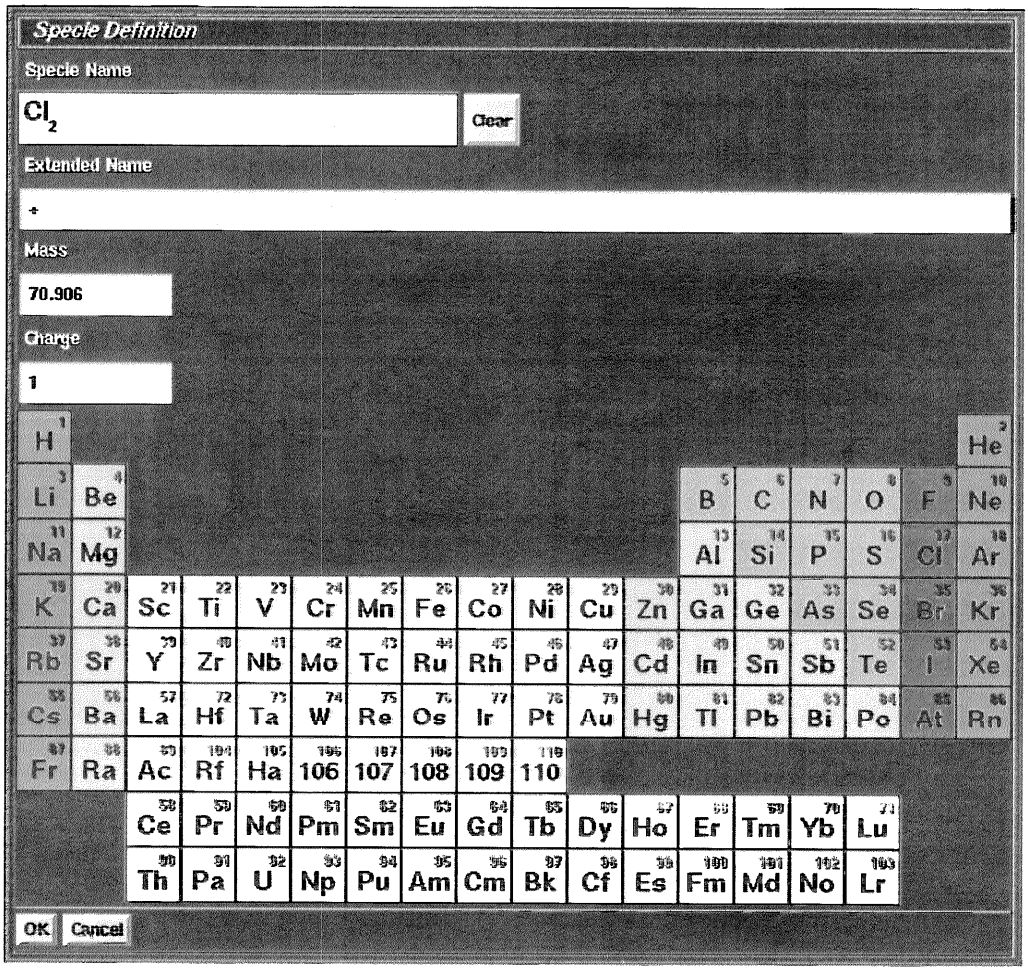

Figure A.13: Species Definition

Figure A.14: Reaction Definition

shown in Figure A.14. Other properties, such as the energy generated by the reaction and the cross-sectional information about the reaction, then define the reaction.

Cross-section information is input in intervals that describe the range over which a particular cross-section description is valid, as shown in FigureA.15. Several different forms for cross-section data are supported, but these are all converted into a universal form for use by *XHawk* . Original data is retained, as well as source information describing the data.

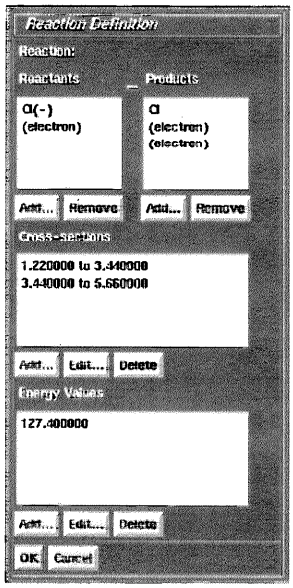

Figure A.14: Reaction Definition

shown in Figure A.14. Other properties, such as the energy generated by the reaction and the cross-sectional information about the reaction, then define the reaction.

Cross-section information is input in intervals that describe the range over which a particular cross-section description is valid, as shown in FigureA.15. Several different forms for cross-section data are supported, but these are all converted into a universal form for use by *XHawk* . Original data is retained, as well as source information describing the data.

Figure A.15: Cross-Section Configuration

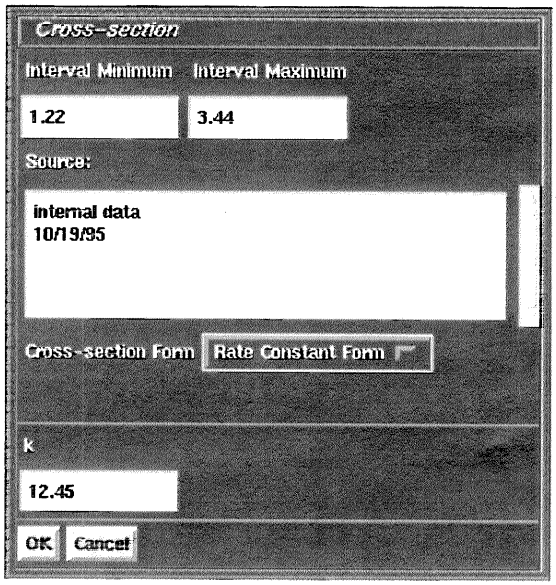

Figure A.15: Cross-Section Configuration

Bibliography

[Alofs71] D. J. Alofs, R. C. Flagan, G. S. Springer. "Density Distribution Measurements in Rarefied Gases Contained between Parallel Plates at High Temperature Differences." *Physics of Fluids.* **14**(3). 1971.

[Anderson84] Anderson, Tannehill, Pletcher. "Computational Fluid Mechanics and Heat Transfer." Hemisphere Publishing Corp. USA, 1984.

[Aydil94] E. Aydil, R. Gottscho, Y. Chabal. "Real-time monitoring of surface chemistry during plasma processing." *Pure and Applied Chemistry.* **66**(6). 1994.

[Aydil95] S. Han, E. Aydil. "A Study of Surface Reactions during Plasma Enhanced Chemical Vapor Deposition of SiO_2 from SiH_4, O_2, and Ar Plasma." *J. Vac. Sci. Technol.*. 1995.

[Bartel95] T. Bartel. "Low Density Gas Modelling in the Microelectronics Industry." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.

[Bird70] G. A. Bird. "Breakdown of Translational and Rotational Equilibrium in Gaseous Expansions." *AIAA Journal.* **8**(11). 1970.

[Bird70a] G. A. Bird. "Direct Simulation of the Boltzmann Equation." *Physics of Fluids.* **13**(11). 1970.

[Bird78] G. Bird. "Simulation of Multi-Dimensional and Chemically Reacting Flows." *Rarefied Gas Dynamics.* 1979.

[Bird76] G.A.Bird. "Molecular Gas Dynamics". Oxford University Press, 1976.

[Bird94] G. Bird. "Molecular Gas Dynamics and the Direct Simulation of Gas Flows." Clarendon Press. Oxford, 1994.

[Boyd91] I. Boyd. "Analysis of vibrational-translational energy transfer using the direct simulation Monte Carlo method." *Physics of Fluids.* **3**(7). 1991.

[Boyd94] I. Boyd, G. Pham-Van-Diep, E. Muntz. "Monte Carlo Computation of Nonequilibrium Flow in a Hypersonic Iodine Wind Tunnel." *AIAA Journal.* **32**(5). 1994.

[Cercignani94] C. Cercignani, R. Illner, M. Pulvirenti. "The Mathematical Theory of Dilute Gases." Springer-Verlag. New York, 1994.

[Dally94] Dally, et. al., "M-Machine Architecture v1.0". Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Concurrent VLSI Architecture Memo 58. 1994.

[Economou91] D. Economou, E. Aydil, G. Barna. "In Situ Monitoring of Etching Uniformity in Plasma Reactors." *Solid State Technology.* April, 1991.

[Foley94] J. Foley. "Introduction to Computer Graphics." Addison-Wesley. Reading, 1994.

[Hildebrand76] F. Hildebrand. "Advanced Calculus for Applications, Second Edition." Prentice-Hall, Inc. Englewood Cliffs, NJ, 1976.

[Hitchon91] W. N. G. Hitchon, T. J. Sommerer, J. E. Lawler. "A Self-Consistent Kinetic Plasma Model with Rapid Convergence." *IEEE Transactions on Plasma Science.* **19**(2). 1991.

[Hitchon94] W. N. G. Hitchon, G. J. Parker, J. E. Lawler. "Accurate Models of Collisions in Glow Discharge Simulations." *IEEE Transactions on Plasma Science.* **22** (3). 1994.

[Hitchon94a] W. N. G. Hitchon, E. R. Keiter. "Kinetic Simulation of a Time-Dependent Two-Dimensional Plasma." *Journal of Computational Physics.* **112** (2). 1994.

[Ivanov88] M. Ivanov, S. Rogasinsky. "Analysis of numerical techniques of the direct simulation Monte Carlo method in the rarefied gas dynamics." *Soviet Journal on Numerical Analysis and Mathematical Modelling.* **2**(6). 1988.

[Ivanov88a] M. Ivanov, S. Rogasinsky, V. Rudyak. "Direct statistical simulation method and master kinetic equation." XVI International Symposium on Rarefied Gas Dynamics, Pasadena. 1988.

[Ivanov91] M. S. Ivanov, S. V. Ragasinsky. "Theoretical Analysis of Traditional and Modern Schemes of the DSMC Method". Invited Paper, Rarefied Gas Dynamics, 1991.

[Ivanov91a] M. Ivanov, S. Rogasinsky. "Theoretical analysis of traditional and modern schemes of the DSMC method." Proceedings of the XVII International Symposium on Rarefied Gas Dynamics, Aachen. 1991.

[Keiter94] E. R. Keiter, W.N.G. Hitchon, M. J. Goeckner. "A kinetic model of pulsed sheaths." *Physics of Plasmas.* **1**(11). 1994.

[Koura92] K. Koura, H. Matsumoto. "Variable soft sphere molecular model for air speciess." *Physics of Fluids.* **4**(5). 1992.

[Legge90] H. Legge, K. Nambu, S. Igarashi. "Force and Heat Transfer on a Disc in Rarefied Flow." *Rarefied Gas Dynamics 17.* Volume 1, Weinheim, NY. 1990.

[Marriott95] P. Marriott, T. Bartel. "Comparisons of DSMC Flow Field Predictions using Different Models for Energy Exchange and Chemical Reaction Probability." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.

[Maskit94] D. Maskit, S. Taylor. "A Message-Driven Programming System for Fine-Grain Multicomputers", *Software - Paractice and Experience.* 24. 1994.

[Nambu89] K. Nambu, S. Igarashi, Y. Watanabe. "Three-Dimensional Hypersonic Flow Around a Disk with Angle of Attack." *Proceedings of the XVI International Symposium on Rarefied Gas Dynamics.* 1989.

[Nambu95] K. Nambu, S. Uchida. "Application of Particle Simulation to Plasma Processing." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.

[Nelson96] B. Nelson, T. Phung, M. Rieffel, S. Shankar, S. Taylor. "Concurrent Plasma Simulations of the GEC Reference Cell Reactor." International Parallel Processing Symposium, Waikiki. 1996.

[Parker94] G. J. Parker, W. N. G. Hitchon, J. E. Lawler. "Numerical solution of the Boltzmann equation in cylindrical geometry." *Physical Review* , **50**(4): 3210-3219. 1994.

[Parker95] G. J. Parker, W. N. G. Hitchon, D. J. Koch. "Transport of sputtered neutral particles." *Physical Review* pre-print, April 1995.

[Parsons95] T. Parsons, J. Harvey. "Object-Process Paradigms in Molecular Computation." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.

[Press92] W. Press, S. Teukolsky, W. Vetterling, B. Flannery. "Numerical Recepies in C." Cambridge University Press. Cambridge, 1992.

[Roy95] R. Samanta Roy, G. Hastings, S. Taylor. "Three-Dimensional Plsma Particle-in-Cell Calculations of Ion Thruster Backflow Contamination." Submitted to *Journal of Computational Physics.*

[Shankar95] S. Shankar, M. Rieffel, S. Taylor, D. Weaver, A. Wulf. "Low Pressure Neutral Transport Modelling for Plasma Reactors." *Invited Paper for 12th International Symposium on Plasma Chemistry*, August 21-25, 1995.

[Taylor95] S. Taylor, J. Watts, M. Rieffel, M. Palmer. "The Concurrent Graph: Basic Technology for Irregular Problems". Submitted to Journal of Comptational Science. 1995.

[Taylor95a] S. Taylor and J. Wang. "Large-scale Simulations of the Delta II Launch Vehicle". Proceedings of Parallel CFD 1995.

[Wadsworth93] Dean C. Wadsworth. "Slip effects in a confined rarefied gas. I: Temperature slip." *Physics of Fluids.* **5**(7). 1993.

[Wadsworth95] D. Wadsworth. "Development and Application of a Three-dimensional Parallel Direct Simulation Monte Carlo Code for Materials Processing Problems." Parallel CFD '95, Pasadena. 1995.

[Watts95] J. Watts. "A Practical Approach to Dynamic Load Balancing." California Institute of Technology Masters Thesis, 1995.

[Watts95a] J. Watts. "The Extended I/O Library." California Institute of Technology Technical Report, 1995.

[Yokokawa91] M. Yokokawa, K. Watanabe, H. Yamamoto, M. Fujisaki, H. Kaburaki. "Parallel Processing for the Direct Simulation Monte Carlo Method." *Computational Fluid Dynamics Journal.* **1**(3). 1992.

[Zhong95] X. Zhong, K. Koura. "Comparison of Solutions of the Burnett Equations, Navier-Stokes Equations, and DSMC for Couette Flow." *Rarefied Gas Dynamics 19.* Volume 1, Oxford University Press. 1995.